

Prix: 590 F par abonnement.

Sommaire de la revue :



Présentation des disquettes ToolBox Mag Numéro 6

Eh oui, on prend ses aises: deux disquettes désormais dans chaque numéro de ToolBox-Mag! Et pour le même prix! On va enfin pouvoir faire confortablement de l'hypermédia… Voici les catalogues:

Disquette 6A :

/ANIM.TBMAG6: la présentation de ce numéro 6. Les graphismes sont d'*Olivier Bailly-Maître,* la musique de *Sébastien Mousseron,* l'animation d'Olivier Goguel. Bootez la disquette!
/ANNONCES: il faut tout lire dans ToolBox-Mag, y compris les petites annonces…
/BABAR: la suite (partie Midi) de l'étude de *Jean-Pierre Charpentier* sur l'outil SynthLab et le "S" de GS. Branchez le DX7 et la boîte à rythmes sur le GS, et chauffe Marcel! Attention, pour que ça marche, mettez le *Tool.Setup* de la disquette à la place du vôtre (bug du 5.04 français). Voyez pages 36-38.
/DERNIERE.HEURE: lisez le fichier *Lisez.Moi.*
/MINIDESKTOP: les sources et l'application qui servent de support aux cours d'initiation à la programmation du GS du *Professeur Destelle.* Voyez pages 39 à 50. Rappel: les fichiers se terminant par *.rsrc* sont des sources pour ResDoctor (disquette B).
/STARTTOOLS.C: c'est *Philippe Manet* qui remplace F. Uhrich pour notre rubrique sur le C de ce numéro 6, avec une version en C de la routine de Startup des outils pour les accessoires de Patrick Desnoues. Voyez pages 10-11.
/SYS.5.04.US: le fichier du système 5.04 US nécessaire pour faire tourner *MuSeq* de J.P. Charpentier. Ce n'est pas notre faute, mais nos excuses quand même.
/TEMPS.TRAVAIL: le "programme à écrire soi-même" de *Patrick Desnoues.* Voyez pages 21-23.
/VISIT.MONITOR: init et source Merlin de *"Visit Monitor II: the Revenge"*, d'*Olivier Goguel.* Voyez pages 4 à 8.
PRODOS est toujours un faux: voyez l'explication page 30.

Disquette 6B :

/PAROLE.PILES: deux piles HyperCard couleur pour faire parler le GS. Voyez l'article de *Jacques Liautard* pages 12-17.
/RESDOCTOR: un éditeur de ressources complet pour le prix d'un numéro de ToolBox-Mag! Avec toute sa documentation sous formes de fichiers texte. Grâce à *Jean Destelle* (voir pages 26-27). Adieu sans regrets à Rez et DeRez…

ToolBox-Mag

Directeur de la Publication
Eric Weyland

Rédacteur en Chef
Jean-Yves Bourdin

Mise en pages
Dominique Digoy

Secrétariat
Janine Loiseleux

Conseil de Rédaction
Jean-Pierre Charpentier
Dominique Delay
Patrick Desnoues
Bernard Fournier
Olivier Goguel
Stéphan Hadinger
François Hermellin
Hubert Loiseleux
Claude Pélisson
Jean-Luc Schmitt
François Uhrich

Rédaction, Siège Social
ToolBox-Mag
6, rue Henri-Barbusse
95100 Argenteuil - France
Téléphone: (1) 30 76 18 64
Télécopie: (1) 39 47 44 08

Impression
Imprimerie /Reprographie /Graphisme - Argenteuil

Les auteurs de ToolBox-Mag n° 6

Olivier Bailly-Maître
Jean-Yves Bourdin
Jean-Pierre Charpentier
Patrick Desnoues
Jean Destelle
Bernard Fournier
Olivier Goguel
Jacques Liautard
Philippe Manet
Sébastien Mousseron
Eric Weyland

L'Editorial du Directeur :

Prenons nos aises

Deux revues et trois disquettes d'un coup dans la même enveloppe, cette livraison va vous donner de l'occupation pour toutes vos vacances! Mais soyons francs: si vous avez reçu dès Juin le numéro d'Août/Septembre, c'est que nous avons besoin d'un peu de temps; pour souffler un peu et pour bronzer, certes, mais surtout pour **faire des projets**.

En effet, ToolBox-Mag arrive à un **tournant important**: le numéro 6, c'est le terme de la première série, **le moment où la plupart d'entre vous ont à se réabonner**. C'est le moment où on fait le point, où on trace des perspectives.

D'ores et déjà, vous pouvez constater quelques changements:

• Le principal: nous mettons un terme aux revendications du Rédac'Chef. ToolBox-Mag, à partir du numéro 6, prend ses aises, et comporte désormais **deux disquettes au lieu d'une!**
• Nous pourrons ainsi nous orienter franchement vers l'**hypermédia** (HyperStudio, et surtout **HyperCard Couleur**).
• **L'initiation à l'Apple II GS** prend de l'ampleur: nous en arrivons à l'initiation à la **programmation** du GS.
• Au plan financier, nous maintenons notre choix: **la qualité du contenu avant tout**, intégralement financée par les abonnements. Rien n'étant gratuit en ce bas monde (sauf la deuxième disquette du numéro 6!), le passage de six disquettes à douze se traduit donc par une augmentation (raisonnable) du tarif de l'abonnement, à compter du numéro 7.

ToolBox-Mag est-il cher?
Prenez votre collection des six premiers numéros: en se bornant aux programmes qu'ils contiennent, **s'il s'était agi de logiciels commerciaux, ou même de shareware, combien auriez-vous payé pour avoir tout cela?**

D'autant que nous avons une **surprise** pour vous: le **réabonnement à l'ancien tarif s'il nous parvient avant le 14 Juillet, soit six disquettes ToolBox-Mag gratuites!** (Pour ne rien vous cacher: Septembre, c'est le "gros coup" de l'Apple-Expo. Nous aimerions bien avoir liquidé la gestion des réabonnements avant les vacances d'Août: d'où ce butoir **absolu** du 14 Juillet).

Grâce à vous, ToolBox-Mag a **gagné son premier pari: exister,** et les utilisateurs de GS ont désormais leur magazine en français. A vous de lui faire **gagner son second pari**: prendre ses aises et son confort, s'étaler un peu, **s'installer** en somme.

S'étaler, s'installer, n'y aurait-il pas dans cette volonté, pour un magazine GS, un peu d'insolence? Ma foi, pas plus que dans notre devise commune: **Apple II fort et vert !**

ERIC WEYLAND

PS: Pensez à lire les petites annonces de la disquette...

Visit Monitor II

Olivier Goguel

Dans l'architecture du GS, l'un des éléments que les vrais programmeurs apprécient le plus est l'accessoire de bureau *Visit Monitor*, qui leur permet à tout moment (ou presque) d'aller explorer et/ou modifier la mémoire de l'ordinateur comme bon leur semble. Ce qui est très pratique pour mettre au point un programme, puisque l'on peut, par exemple, patcher certaines routines ou changer directement certaines variables sans avoir besoin de réassembler (*Les vrais programmeurs ne compilent pas; ils assemblent !*).
Malheureusement, ce que ne donne pas Visit Monitor (et tous les clones qui ont été écrits jusqu'à présent: *Nifty List, Diversi-Hack…*), c'est le compteur ordinal (PC=Program Counter), c'est-à-dire l'adresse où le programme a été interrompu par l'activation du Control-Panel. Connaître le PC est une aide importante pour les vrais programmeurs, qui peuvent suivre plus facilement l'exécution des programmes; et c'est ce que l'on se propose de connaître à travers le programme Visit Monitor II™.

Mise en œuvre de Visit Monitor II™

Elle est des plus aisées... Il suffit de copier le fichier VMonitor.II (de type PIF=$B6) dans le sous-catalogue */System/System.Setup* de votre disque système.
Lors du boot, une icône apparaît en bas à gauche pour vous indiquer que Visit Monitor II™ est actif. Ensuite, lors de votre prochain accès au Control-Panel,Visit Monitor II™ est installé et utilisable à la place de Visit Monitor (qui n'apparaît plus dans la liste des CDA).

Si vous choisissez Visit Monitor II™, vous arrivez dans le moniteur (Ctrl-Y ou Q pour ressortir) et les registres PC, B, D, m, x et e sont affichés. Les autres registres sont accessibles par la commande moniteur Ctrl-E.
De plus, les pointeurs du moniteur sont automatiquement positionnés sur le PC: de ce fait, si on tape L par exemple, on désassemble directement la zone pointée par le PC avec les bons modes de désassemblage.

Principaux problèmes liés à la programmation de Visit Monitor II™

Tout d'abord, on a besoin de **connaître le PC**, c'est notre but originel.
Le PC se trouve, au moment du processus d'interruption, empilé ainsi que le registre d'état (P). Ensuite, le gestionnaire d'interruption (Interrupt Manager, qui est susceptible de varier selon les Roms) est exécuté: il va lancer la routine correspondant à la nature de l'interruption.
Ces deux niveaux de routines vont, à leur tour, ajouter des informations sur la pile, et on risque d'avoir des difficultés par la suite à retrouver l'adresse exacte du PC.
Afin d'être sûr de ne pas la perdre, on va **directement réécrire le début de l'Interrupt Manager**, pour sauvegarder tout de suite le PC ▶

et les autres registres qui nous intéressent.

Autre problème, puisque Visit Monitor et Visit Monitor II™ ont exactement la même utilité (à part bien sûr l'affichage du PC), il est inutile d'encombrer la liste des accessoires de bureau actifs par ces deux CDA; on va donc **remplacer Visit Monitor par Visit Monitor II™.**

Ces deux problèmes vont engendrer la transformation de notre fichier VMonitor.II, que l'on pensait au début générer en tant que CDA, en fichier PIF, c'est-à-dire Permanent Initialisation File. En effet, on va avoir besoin, avant de pouvoir exécuter Visit Monitor II™, d'installer notre nouvel Interrupt Manager ainsi que de remplacer Visit Monitor par Visit Monitor II™.

En ce qui concerne la création d'un fichier de type PIF, cela ne pose absolument aucun problème (du moins en assembleur, si j'ai bien compris ToolBox-Mag 4...); un init PIF est un fichier GS/OS, qui est exécuté dès son chargement par le système. Voyez ToolBox-Mag 4 sur les inits (PIF et TIF).
On inclura donc dans une première partie de ce fichier les routines permettant d'installer le nouvel Interrupt Manager et de remplacer Visit Monitor par Visit Monitor II.
Ensuite, dans une deuxième partie, on inclura le nouvel Interrupt Manager ainsi que le CDA Visit Monitor II.

Dernier problème, purement décoratif : **afficher une icône** lors de l'installation de l'init. Il n'existe aucune fonction permettant de faire cela automatiquement. Seul le Resource Manager le permettrait, mais je ne suis pas (au contraire de mon néanmoins ami Bernard) un fan des ressources; on va, comme d'habitude, se débrouiller tout seul.
On va simplement afficher une icône directement sur l'écran du GS à un emplacement défini. *Cet emplacement pourra être très facilement modifié dans le source s'il correspond déjà à une icône de votre système.*
Pour simplifier la routine d'affichage et la place de l'init, j'affiche une icône parfaitement rectangulaire, ce qui ne m'oblige pas à gérer un éventuel masque d'affichage.

Bibliographie: voyez le "GS Epluché" si les mots "PC", "registres", etc, ne sont pas clairs pour vous.

Visit Monitor II v1.01

(c) FTA & ToolBox-Mag, Mars 1991
Assemblage par OA-6 sous Merlin16+

```
            REL
            TYP   $B6           Fichier PIF
            DSK   VMONITOR.II.L
QAdr        =     $0003D0
UsrAdr      =     $0003F8       Vecteurs de retour
Routine d'initialisation
Start       PHP
            SEI
            REP   $30
            PHB
            PHK
            PLB
            TSC                 Rend disponibles
            SEC                 les $10 premières
            SBC   #$10          valeurs de la page Zero
            TCS
            PHD
            INC
            TCD
            JSR   InstallVMII   Installe VMonitor II
            BCS   Not
            JSR   DrawWindowAffiche l'icône de l'init
            LDX   #2
]lp         LDAL  $E10010,X     Remplace l'Interrupt
            STA   vector_rom,XManager
            LDA   NewVect,X
            STAL  $E10010,X
            DEX
            DEX
            BPL   ]lp
Not         PLD                 Retour propre
            TSC                 au programme appelant
            CLC
            ADC#$10
            TCS
            PLB
            PLP
            RTL
Nouvelle routine d'interrupt manager...
int_manager CLC
            XCE
            PHP
            REP   #$30
            SEI
            PHA
            PHB
            PHK
            PLB
            LDAL  $E1C068
            PHA
            AND   #$FFCF        Elimine une éventuelle
            STAL  $E1C068       autorisation de lecture en
                                bankI, sinon, on lira
                                $E11D66 au lieu de $E01D66
Regarde si on est déjà dans le control-panel
            LDAL  $E01D66       CPanel
            CMP   #$FF00        C=0
            PLA
            STAL  $E1C068       Si Oui, les nouvelles
                                valeurs des registres
            BCC   NoSaveAdr     ne nous intéressent pas
Si non, on va sauvegarder les registres qui nous semblent
utiles. A ce niveau-là, la pile se présente comme suit :
* 6       PC   (Adr)   Empilé automatiquement par
* 5       P    (Byte)  le mécanisme d'interrupt manager
* 3       A    (Word)  Empilé par notre interrupt manager
* 1       B    (Byte)  idem
          SP   ->
```

▶

```
        LDAL $E1C068
        STA  Sw68
        LDAL $E1C035
        STA  Sw35

        LDA  $02,S
        STA  AAdr
        STY  YAdr
        STX  XAdr
        LDA  $01,S
        STA  BAdr
        TDC
        STA  ZPageAdr
        LDA  $06,S
        STA  PCAdr
        LDA  $04,S
        BIT  #%0000_0000_0000_0001
        BEQ  NoForce_xm
        ORA  #%0011_0000_0000_0000
NoForce_xm STA PAdr
        AND  #%0000_0000_0000_0001
        BEQ  E0
        LDAL $E1C068      Mode Emulation
        BIT  #$0010
        CLC               Force C=0
        PHP
        LDA  $08,S
        AND  #$00FF       On est soit en bank0
        PLP
        BEQ  PC1
        ORA  #$0100       soit en bank1 si on travaille
        BRA  PC1          dans la mem aux (bit4 C068)
E0      LDA  $07,S        Mode Natif
        SEC               C=1
PC1     STA  PCAdr+1
        TSC
        ADC  #7           Ajoute 7 ou 8
        STA  StackAdr     suivant que l'on est
                          en mode emulation ou non
NoSaveAdr PLB
        PLA
        PLP
        XCE
vector_rom JML $000000    et saute au manager
                          initial en rom

NewVect JML int_manager   Nouveau vecteur
                          de l'interrupt manager


        MX   %00
```

Installation de VMonitor II

```
InstallVMII LDY #0
        JSR  FEB4         Installe Visit Monitor
                          au cas où il serait absent
```

Maintenant, on va tracer tous les CDA jusqu'à ce que l'on trouve Visit Monitor

```
        LDAL $E01D84      Handle sur les handles
        STA  $00          des CDA
        LDAL $E01D86
        STA  $02

        LDY  #2           on le déréference
        LDA  [$00]
        TAX
        LDA  [$00],Y
        STA  $02
        STX  $00

        STZ  $08          Compteur de CDA

SearchNext LDY #2         On prend le handle
        LDA  [$00]        du CDA référencé
        STA  $04
        LDA  [$00],Y
        STA  $06

        LDA  [$04]        On le déréference
        TAX
        LDA  [$04],Y
        STA  $06          Et on tombe directement
        STX  $04          sur le nom du CDA
```

On regarde si l'on a trouvé le CDA "Visit Monitor"

```
        LDA  [$04]        On regarde si la
        CMP  #$D60D       longueur du nom = $D
        BNE  NotFound     et si la premiere lettre = "V"
```

Si oui, on va tester les autres lettres en utilisant la chaîne "Visit Monitor" incluse dans "Visit Monitor II" pour effectuer la recherche...

```
        LDY  #2
]lp     LDA  [$04],Y
        CMP  SearchString,Y
        BNE  NotFound
        INY
        INY
        CPY  #14
        BNE  ]lp
```

On a enfin trouvé le CDA visit Monitor. On va l'éliminer de la liste des CDA en remplacant son handle par un handle sur notre CDA : Visit Monitor II

```
        LDY  #2
        LDA  #VisitMonitorII
        STA  [$00]
        LDA  #^VisitMonitorII
        STA  [$00],Y
        CLC
        RTS               Installation terminée
```

Le CDA trouvé ne s'appelait pas Visit Monitor, on va tester le suivant...

```
NotFound INC  $08
        LDA  $08
        CMPL $E01D7C      Nb de CDA Actifs
        BCC  Again
```

Aucun CDA ne s'appelle Visit Monitor, ce qui est en théorie impensable, (à moins que VM II ne soit déjà installé...).

```
        SEC               (optionnel)
        RTS               Sortie avec erreur

Again   LDA  $00          Passage au handle
        CLC               suivant...
        ADC  #4
        STA  $00
        BRA  SearchNext
```

CDA Visit Monitor II : The Revenge (!)
largement inspiré du source du CDA Visit Monitor d'Apple...

```
VisitMonitorII ADRL *+4   Ptr sur le CDA

SearchString STR "Visit Monitor II" Nom du CDA

        ADRL LaunchMonitor Adresse d'entrée
        ADRL QuitMonitor   Adresse de Sortie,
                           pointe sur un RTL.

LaunchMonitor SEP $30
        SEI
        PEA  $0000
        PLB               B=0
        PLB

        LDX  #2           Sauvegarde dans la pile
]lp     LDA  UsrAdr,X     l'ancienne adresse du
        PHA               vecteur ctrl-y et de
        LDA  QAdr,X       du vecteur "Q"
        PHA
        LDAL NewUsr,X
        STA  UsrAdr,X     Met l'adresse de
        STA  QAdr,X       retour au CDA...
        DEX
        BPL  ]lp

        LDA  #0
        LDX  #<$FF69
        ROL  $C025        Si on appuie sur ,
        BCS  AppleVM      on a exactement
        LDA  #"M"&$9F     le Visit Monitor d'Apple
        LDX  #<$FF69+3    (ou presque...)
AppleVM STAL PatchText
        TXA
        STAL PatchJmp+1
```

▶

```
        LDX   #8
]lp     LDA   $F7,X        Sauvegarde et remplace
        PHA                la zone $F7 à $FF par
        LDAL  MoveRout,X   les points d'entrée
        STA   $F7,X        et de sortie de VM II
        DEX
        BPL   ]lp
        REP   $30
        JSR   FC58         Efface l'écran
        LDY   #0           Affichage du texte
        TYX                et des différents
]lp     LDAL  Text_II,X    registres concernés
        AND   #$00FF
        BEQ   EndText
        JSR   FDED
        CMP   #"="
        BNE   NoPrintValue
        PHX
        TYX
        JSR   (SpecialRout,X)
        INY
        INY
        PLX
NoPrintValue INX
        BRA   ]lp
EndText SEP   $30
        SEC                Exécution du moniteur
        XCE
        JML   $0000F7
```

Retour après CTRL-Y

```
RetourAdr PLA              On remet la Pile
        PLA                à un niveau honorable
        PEA   $0000
        PLB
        PLB
        CLC
        XCE
        CLI
        LDX   #$00
]loop   PLA
        STA   $F7,X        On remet les adresses
]lp     INX                que l'on avait patchées
        CPX   #$09
        BCC   ]loop
        PLA
        STA   QAdr-9,X
        PLA
        STA   UsrAdr-9,X
        CPX   #$0B
        BCC   ]lp
        REP   $30
QuitMonitor RTL            Retour à l'appelant,
                           càd le gestionnaire de CDA
NewUsr  JMP   $00FB        Nouveau vecteur pour Ctrl-Y
```

Routines déplacées en $00F7 pour faire le lien entre le CDA et le moniteur

```
MoveRout CLI
PatchJmp JMP  $FF69+3      +3 pour éviter l'initialisation
                           des variables registres du
                           moniteur
        SEI
        JML   RetourAdr
```

Routines permettant d'appeler des fonctions standard du moniteur depuis n'importe quel banc...

```
FEB4    PEA   $FEB4
        BRA   Emulation
FDED    PEA   $FDED
        BRA   Emulation
FC58    PEA   $FC58
        BRA   Emulation
FDDA    PEA   $FDDA
Emulation PHA
        PHA
        PHA
        PHA
        PHX
        PHY
        LDA   $0D,S
        PHA
        LDX   #$2403       _FW Entry
        JSL   $E10000
        PLY
        PLX
        PLA
        PLP
        PLP
        RTS
Text_II ASC   "Visit Monitor II by FTA.",8D
PatchText HEX 8D
        ASC   " PC= B= D= m= x= e=",00
```

Affichage des différents registres...

```
SpecialRout DA PrintPC
        DA    PrintB
        DA    PrintD
        DA    Printm
        DA    Printx
        DA    Printe
PrintPC LDX   #2
]lp     LDAL  PCAdr,X
        JSR   FDDA
        DEX
        BPL   ]lp
        JMP   MoveParms    Envoie les registres
                           aux variables du moniteur.
PrintD  LDAL  ZPageAdr+1
        JSR   FDDA
        LDAL  ZPageAdr
        JMP   FDDA
PrintB  LDAL  BAdr
        JMP   FDDA
Printm  LDAL  PAdr+1
        BIT   #%0010_0000
TestB   BNE   Off
On      LDA   #"0"
        JMP   FDED
Off     LDA   #"1"
        JMP   FDED
Printx  LDAL  PAdr+1
        BIT   #%0001_0000
        BRA   TestB
Printe  LDAL  PAdr
        BIT   #%0000_0001
        BRA   TestB
```

Zone de stockage des registres

```
AAdr    DS    2
XAdr    DS    2
YAdr    DS    2
StackAdr DS   2
ZPageAdr DS   2
PAdr    DS    2
BAdr    DS    2
PCAdr   DS    4
Sw68    DS    2
Sw35    DS    2
```

Envoie les registres aux variables registres du moniteur

```
MoveParms PHB
        PEA   $E1E1
        PLB
        PLB
        LDX   #8
]lp     LDAL  AAdr,X
        STA   $0120,X
        DEX
        DEX
        BPL   ]lp
        LDAL  PCAdr
        STA   $013A
        LDX   #10
]lp     STA   $3A,X
        DEX
        DEX
        BPL   ]lp
```

▶

```
        SEP   $20
        LDAL PCAdr+2
        STA   $012C       =K
        STA   $013E
        STA   $01A7
        LDAL BAdr
        STA   $012B       =B
        LDAL PAdr+1
        STA   $012A       =P
        LDX   #0
        PHA
        BIT   #%0010_0000
        BEQ   *+3
        INX
        TXA
        STA   $0130       =m
        LDX   #0
        PLA
        BIT   #%0001_0000
        BEQ   *+3
        INX
        TXA
        STA   $0131       =x
        LDAL PAdr
        AND   #%0000_0001
        STA   $0132       =e
        LDAL Sw68
        STA   $012D       =M
        AND   #4
        LSR
        LSR
        STA   $012F       =L
        LDAL Sw35+1
        ASL
        PHP
        LDAL Sw35
        ASL
        PLP
        ROR
        STA   $012E       =Q
        LDA   #0
        STA   $0133       =d
        REP   $20
        PLB
        RTS
```

Affichage de l'icône...

```
DrawWindow LDAL $E1C029
        ORA   #$40
        STAL  $E1C029
        LDY   #0
        TYX
]lp2    SEP   $20
        LDA   #IconNbCol
]lp     PHA
        LDA   WindowAdr,Y
        STAL  $E12000+IconPos,X
        INY
        INX
        PLA
        DEC
        BNE   ]lp
        REP   $20
        TXA
        CLC
        ADC   #$A0-IconNbCol
        TAX
        CPX   #IconNbLgn*$A0
        BCC   ]lp2
        RTS
```

IconeData

```
IconPos   =     169*$A0+5
IconNbLgn =     $0018
IconNbCol =     $0011

WindowAdr HEX ...
```

[voir les data de l'icône dans le fichier source sur le disque]

Accessoires et ressources

Philippe Manet

Quelques remarques à propos de l'article de Bernard Fournier sur les ressources dans ToolBox-Mag 4. La méthode proposée pour réouvrir en lecture-écriture le Resource Fork de l'application me paraît plus compliquée que nécessaire, et ne fonctionnant pas dans certains cas:

• La routine de GS/OS *GetName* ne renvoie que le nom du fichier sans aucun préfixe. Si le préfixe courant ne correspond pas à celui du programme appelant (par exemple si on est dans un NDA), la réouverture du Resource Fork ne trouvera pas le fichier. Il faut utiliser à la place l'une des routines du System Loader: *LGetPathname* (qui renvoie une PString) ou *LGetPathname2* (qui renvoie une Class1 String). C'est d'ailleurs la technique recommandée par le "Bureau Politique" dans la note technique 83.

• L'appel à *OpenResourceFile* avec un premier paramètre à 0 demande à GS/OS d'ouvrir le fichier avec le mode défini par le champ *access* dans le directory. Ainsi, si le fichier a été verrouillé, il ne sera ouvert qu'en lecture: ce n'est peut-être pas l'effet voulu. Il est sans doute préférable de spécifier explicitement le mode lecture/écriture lors de l'écriture, en passant la constante *ReadWriteEnable*, qui vaut 3.

La séquence de réouverture du Resource Fork devient ainsi, plus simplement:

OpenResourceFile (ReadWriteEnable, NIL, LGetpathname2 (UserId,&)).

De plus, l'appel à *UpdateResourceFile* est inutile, car il est fait implicitement par *CloseResourceFile*, dès qu'une ressource a été marquée modifiée par *MarkResourceChange*.

NDLR: Philippe Manet et... Apple ont raison. Ce bon vieux Bureau Politique a tiré une épine du pied de beaucoup de programmeurs avec ses dernières notes. Voyez les contributions de Bernard Fournier dans ToolBox-Mag 5 et 6, et comment Jean Destelle s'est également battu avec ce problème pour Sélect.
Ceci dit, où, en-dehors des Macros et des Includes d'Apple, trouve-t-on une documentation sur ces appels du Loader?

Des ressources dans les accessoires

Bernard Fournier

La *note technique Apple n°71* contient un paragraphe expliquant comment employer des ressources avec les accessoires de bureau (NDA). *A priori*, si une note technique décrit ce point de programmation, on peut raisonnablement supposer qu'il y a un problème pour qui aborde la chose sans ces précisions.

En effet, la programmation d'un NDA avec des ressources va mettre en évidence le problème suivant: **comment indiquer le chemin d'accès originel (le préfixe où se trouve le NDA et ses ressources) à notre NDA pour qu'il puisse correctement charger ses ressources, même si le NDA ne se trouve pas dans le sous-catalogue /DESK.ACCS** (par exemple si on a lancé le NDA à partir du Fonte-DA Installateur de F. Uhrich)?

Ce problème étant particulièrement ardu, Apple a jugé bon de clarifier les choses en expliquant comment procéder. Ceci afin d'éviter des programmations 'sauvages' qui produiraient des désastres.

Dans la routine *OPEN* du NDA, il faut impérativement suivre la séquence suivante:

• faire un *GetCurResourceApp* et conserver le résultat dans une variable

• si on n'a pas déja alloué un ID à notre NDA, faire alors un *MMStartUp*

• faire un *ResourceStartup* avec l'identifiant fraîchement défini. Il est en effet du ressort de chaque application ou NDA de s'assurer que le Resource Manager est correctement initialisé.

• grâce à *LGetPathname2*, on va récupérer un pointeur sur le préfixe d'accès au NDA (passer l'ID défini et la valeur $0001 comme paramètres).

• afin d'éviter la fermeture accidentelle du fichier ressource, on va récupérer le *file level* avec la fonction *GetLevel* et on va le fixer temporairement à 0 avec *SetLevel*.

• ensuite on récupère la valeur des préférences GS/OS par un *GetSysPref*: cette valeur est conservée en lieu sûr, et avec une variable temporaire on va calculer la valeur qu'il faut pour forcer le système à réclamer l'insertion du disque originel, en cas d'éjection intempestive de la disquette contenant les ressources du NDA. Cette valeur s'obtient en faisant un *AND $1FFF* puis un *ORA $8000* à la valeur 'préférences', et on passe ce paramètre à *SetSysPrefs*.

• puis on ouvre le fichier ressource avec *OpenResourceFile* en lui indiquant le préfixe d'accès obtenu plus haut. Conserver soigneusement le *FileID* retourné pour pouvoir fermer le fichier ressources ultérieurement.

• maintenant on remet les valeurs originelles: *SetSysPrefs* avec la valeur conservée; puis *SetLevel* pour remettre le niveau original et enfin un *SetCurResourceApp* pour remettre définitivement les choses en l'état.

• dans la routine *Close*, on aura deux choses à faire: un *CloseResourceFile* avec le *FileID* retourné lors de l'ouverture; puis un *ResourceShutDown* afin de remettre les choses en l'état.

Voici ce que donnent ces consignes une fois transcrites:

dans la routine *OPEN*:

```
OrgApp:=GetCurResourceApp;
MyNdaID:=MMStartUp;
ResourceStartUp(MyNdaID);
PathPtr:=LGetPathName2(MyNdaID,$0001);
OrgLevel:=GetLevel;
OrgPref:=GetSysPref;
NDAPref:=Band(OrgPref,$1FFF);
NDAPref:=Bor(OrgPref,$8000);
SetSysPref(NDAPref);
ResID:=OpenResourceFile($C3,nil,PathPtr);
SetSysPrefs(OrgPref);
SetLevel(OrgLevel);
SetCurResourceApp(OrgApp);
```

dans la routine *CLOSE*:

```
CloseResourceFile(ResID);
ResourceShutDown;
```

Et voilà, muni de ces renseignements, vous devriez pouvoir construire aisément vos NDA avec des ressources. On se donne rendez-vous dans ToolBox-Mag n° 7 pour un listing complet de NDA avec ressources? D'ici là, essayez de construire vos routines, envoyez-les à la rédaction.

Et bonnes vacances! ■

Traduction :

StartTools en C

Philippe Manet

NDLR: Dans notre série des "traductions", voici une version en C, écrite par Philippe Manet, de la routine de StartUpTools pour les NDAs publiée en assembleur par Patrick Desnoues dans ToolBox-Mag 1. Vous trouverez les fichiers correspondants (StartTools.CC et StartTools.h) sur la disquette.
Non seulement vous écrirez plus facilement vos accessoires en C, mais les sources donnent un exemple d'une instruction fort intéressante du C, dont nos spécialistes vous entretiendront un jour: la fonction… ASM!

StartTools.cc

[Démarrage et arrêt d'outils pour les NDA, reprenant la fonction de StartUpTools du système 5.0 qui ne marche que pour les applications.
Ces routines sont basées sur une table qui décrit les outils à démarrer; cette table est mise à jour pendant l'exécution de la routine de démarrage pour la routine d'arrêt.
Adaptation en C des routines assembleur de Patrick Desnoues publiées dans ToolBox-Mag numéro 1.]

```
#pragma optimize -1

#include <Types.h>
#include <Locator.h>
#include <Memory.h>
#include <Resources.h>

#include <orca.h>

#define _StartTools
#include "StartTools.h"
#undef _StartTools

[StartNDATools () - Démarrage des outils définis dans la
liste. Le Resource Manager est démarré d'office.]
Word StartNDATools ( Word userID, Word numTools,
StartToolsPtr toolTable )
{
   Word error, toolNum, toolStatus, parmFlags;
   long numDPages; Pointer DPAddr;
   StartToolsPtr toolEntry;
   [ResManager est démarré systématiquement].
   ResourceStartUp ( userID );
   if ( error = toolerror () )
      return ( error );
   [Pour chacun des outils de la table, on teste son état et,
   en fonction du résultat, on le charge et on le démarre.]
   for ( toolEntry = toolTable; numTools--; toolEntry++ ) {
      toolNum = toolEntry->toolNumber;
asm {     [appel de la fonction status de l'outil]
   lda #0
   pha
   clc
   lda toolNum
   adc #StatusCall
   tax
   jsl ToolBoxEntry
   sta error
   pla
   sta toolStatus
}
if ( error ) {       [l'outil doit être chargé]
   LoadOneTool ( toolNum, toolEntry-
   toolVersion );
   if ( error = toolerror () )
        return ( error );
   toolEntry->status = ToolLoaded;
} else
   toolEntry->status = toolStatus;
[Initialisation de l'outil après allocation éventuelle
des pages directes requises]
if ( ! toolStatus ) {
   parmFlags = toolEntry->parmFlags
   & numParmsMask;
   if ( parmFlags & DPParm ) {
        numDPages = ( toolEntry->parmFlags
        & numDPMask ) >> DPOffset;
        DPAddr = * NewHandle (numDPages,
        userID | DPAuxID,attrLocked + attrFixed
        + attrPage + attrBank, NULL );
        if ( error = toolerror () )
           return ( error );
   }  [allocation de la page directe]
   if ( parmFlags & userIDParm )
        asm {
           pei userID
           }
   if ( parmFlags & DPParm )
        asm {
           pei DPAddr
```

▶

```
        }
    asm {  [appel de la fonction StartUp de l'outil]
        clc
        lda toolNum
        adc #StartUpCall
        tax
        jsl ToolBoxEntry
        sta error
        }
    if ( error = toolerror () )
        return ( error );
  } [outil à initialiser]
 } [boucle chargement/démarrage
   des outils de la table]
 return ( 0 );         [pas d'erreur]
} [StartNDATools ()]
```

[StopNDATools () - Arrêt des outils démarrés précédemment]

```
void StopNDATools ( Word userID,
Word numTools, StartToolsPtr toolTable )
{
  StartToolsPtr toolEntry;
  Word toolNum, toolStatus;
  for ( toolEntry = toolTable; numTools--; toolEntry++ ) {
    toolNum = toolEntry->toolNumber;
    toolStatus = toolEntry->status;
    if ( ! toolStatus || toolStatus == ToolLoaded )
      asm {  [appel de la fonction ShutDown de l'outil]
        clc
        lda toolNum
        adc #ShutDownCall
        tax
        jsl ToolBoxEntry
        }
    if ( toolStatus == ToolLoaded )
      UnloadOneTool ( toolNum );
  } [Boucle d'arrêt de tous les outils démarrés]
  DisposeAll ( userID | DPAuxID ); [Libération de la
  mémoire des pages directes]
  ResourceShutDown ();     [arrêt du Res Manager]
} [StopNDATools ()]
```

StartTools.h

[Fichier à inclure par toute application amenée à utiliser les routines de StartTools.CC]

```
#ifndef __StartTools__
#define __StartTools__

#ifndef __TYPES__
#include <Types.h>
#endif

#define ToolLocator        0x01
#define MemoryManager      0x02
#define Miscellaneous      0x03
#define QuickDraw          0x04
#define DeskManager        0x05
#define EventManager       0x06
#define Scheduler          0x07
#define SoundTool          0x08
#define ADB                0x09
#define Sane               0x0A
#define IntegerMath        0x0B
#define TextTool           0x0C
#define WindowManager      0x0E
#define MenuManager        0x0F
#define ControlManager     0x10
#define SystemLoader       0x11
#define QDAuxiliary        0x12
#define PrintManager       0x13
#define LineEdit           0x14
#define DialogManager      0x15
#define ScrapManager       0x16
#define StandardFile       0x17
#define NoteSynthetize     0x19
#define NoteSequencer      0x1A
#define FontManager        0x1B
#define ListManager        0x1C
#define ACE                0x1D
#define ResourceManager    0x1E
#define MidiTool           0x20
#define TextEdit           0x22
```

[Définitions spécifiques aux routines de démarrage/arrêt]

```
#ifdef _StartTools
#define ToolBoxEntry       0xE10000
#define StartUpCall        0x0200
#define ShutDownCall       0x0300
#define StatusCall         0x0600
#define numParmsMask       0x0003
#define numDPMask          0x0300
#define DPOffset           8
#define ToolLoaded         0xB123     [indique que
l'on a fait LoadOneTool]
#define DPAuxID            0x0F00     [auxID pour la
mémoire allouée pour les pages directes]
#endif
```

[Valeurs des flags du champ parmsFlags ci-dessous]

```
#define userIDParm         0x0001
#define DPParm             0x0002
#define oneDP              0x0100
#define twoDP              0x0200
#define threeDP            0x0300
```

[Cette structure décrit un outil à charger et démarrer]

```
typedef struct {
  Word toolNumber;         [numéro de l'outil]
  Word status;             [0 = inactif, 0xB123
   inactif charge, x = actif]
  Word parmFlags;          [0 = aucun, 1 = userID, 2 =
   Direct Page, 3 = userID + DP, 0x0?00 = nb de DP]
  Word toolVersion;        [version minimum requise]
} StartToolsRec, *StartToolsPtr, *StartToolHndl;
```

[Prototypes des fonctions disponibles :]

```
Word StartNDATools ( Word userID, Word numTools,
StartToolsPtr toolTable );
void StopNDATools ( Word userID, Word numTools,
StartToolsPtr toolTable );
#endif
```

Il ne lui manque même pas la parole !

Jacques Liautard

Quand tout le monde n'en était encore qu'à l'Apple IIE, qui ne rêvait pas de faire parler son ordinateur? Quelques programmes proposaient une possibilité de rentrer par la prise K7 un son qui semblait, quand on arrivait à le ressortir, être le fin du fin... Les cartes Porte-parole tentaient aussi de réaliser le rêve, mais n'étaient pas à la portée de toutes les bourses.

Avec le II GS, le monde du son s'ouvre à tous. Je ne parle pas ici des possibilités de sortie, soit par la sortie haut parleur, soit par une carte stéréo, mais plutôt de comment faire reconnaître un son, que ce soit en parole de synthèse, ou en son digitalisé.

Parole de synthèse et son digitalisé

Quelle est la différence fondamentale entre ces deux types de sons? Eh bien, la même qu'entre un dessin tracé sous Platinum Paint et une image digitalisée avec Quickie. Soit on "trace" soi-même, soit on réduit à des nombres des données extérieures.
Concrètement, deux différences importantes: en premier lieu, la méthode employée, en second lieu, la taille du fichier obtenu.

Les méthodes sont effectivement fondamentalement différentes. La parole de synthèse fait appel à la programmation et uniquement à la programmation, pas besoin de rajouter une carte particulière. Le logiciel *TML Speech Toolkit* suffit.
La parole digitalisée, quant à elle, demande une carte. *Sonic Blaster* ou *Audio Animator* sont les plus diffusées, on peut parler aussi de la carte livrée avec le programme *HyperStudio*, qui est en mono, mais suffit pour la parole digitalisée.

La parole de synthèse

Principaux avantages de la parole de synthèse, la taille du fichier à utiliser est minime et surtout, elle permet un dialogue aléatoire avec un utilisateur. Alors est-ce la panacée? Non, pas vraiment, car son défaut est d'avoir été créée aux Etats-Unis, avec près de 1200 règles grammaticales et du coup de parler en anglais... Mais, rassurez vous, il est possible de l'utiliser en faisant parler en français, en espagnol, en italien, en allemand etc, ceci au prix de quelques modifications ou astuces dont nous parlerons un peu plus loin.

La parole interactive

Comment utiliser la parole de synthèse de manière interactive, c'est à dire en réalisant un dialogue, et donc sans parler des programmes comme la série des *MATH TALK, FIRST LETTERS AND WORDS, FIRST SHAPES*, etc, programmes pour les petits dont les paroles sont entièrement de synthèse, mais qui ne vous permettent pas de taper vos propres mots?
Deux possibilités s'offrent à vous. En programmant ou sans programmer.

Les programmes existants

Sans passer par la programmation, nous trouvons deux traitements de texte du commerce : *SMOOTHTALKER* et *KIDSTALK*, (qui ne diffère de *SMOOTHTALKER* que par l'environnement plus adapté aux enfants), le programme d'exemple livré avec *TML SPEECH TOOLKIT* : *LET'S TALK*, et un freeware (de votre serviteur): *SAKOZ*, destiné en même temps aux enfants en leur permettant d'entendre les lettres ou chiffres qu'ils tapent au clavier, et aux adultes car une boîte de dialogue permet de jongler avec les phonèmes, et les sources en C, présents sur la disquette, permettent de s'initier à la programmation de la parole de synthèse.

En programmant soi-même

Le vrai domaine de la parole de synthèse reste la programmation. Vous désirez établir un dialogue sonore avec l'utilisateur, aucun problème, à chaque lettre, symbole ou mot tapé, le son correspondant sortira sur le haut parleur.
En mode programmation, il ne faut pas oublier d'inclure les *TOOLS 50*, *51* et *52* dans le sous-catalogue */Tools* de votre système.

▶

Quels sont les langages comportant des librairies pour utiliser ces outils? *TML BASIC*, le *PASCAL* et le *C*. Donc vous avez l'embarras du choix, à chacun sa librairie.

Francisation de la parole de synthèse

Oui, mais, que ce soient les traitements de texte ou les outils de programmation, c'est prononcé en anglais !
Effectivement, nous en arrivons aux trucs et astuces pour faire parler le II GS en bon français, ou plutôt dans un français avec un accent anglais plus ou moins prononcé suivant les mots.

La parole de synthèse utilise la phonétique. Les phonèmes sont les éléments sonores du langage, qui, combinés, nous font entendre un son qui correspond à une lettre, mot ou phrase. Le *TML SPEECH TOOLKIT* utilise les 41 codes phonétiques suivants :

Æ - EH - IH - AA - AH - EY - IY - AY - OW - UW - AO - UH - AX - OY - AW - ER - p - t - CH - k - b - d - j - g - f - TH - s - SH - v - DH - z - ZH - l - m - n - NG - h - r - w - WH - y

Attention à surtout bien respecter les majuscules/minuscules, sinon les phonèmes ne seront pas prononcés.

Si vous utilisez un des programmes mentionnés plus haut, vous pouvez visualiser quel phonème anglais correspondant à quelle lettre. Par exemple, le A aura pour phonème AX et le E pour phonème IY. C'est là qu'intervient une astuce, pourquoi ne pas aller voir un peu à quoi ressemble le programme *SMOOTHTALKER* ou bien le *TOOL052*...

Comment? Avec Appleworks 3.0 par exemple, pour déjà avoir une idée de ce que l'on recherche, puis avec un éditeur de blocs pour faire quelques modifications.
Chargez avec Appleworks 3.0, en fichier texte ascii, le programme SmoothTalker, ou le Tool052.
Une fois sur le bureau, faîtes une recherche de AX dans Smoothtalker par exemple, vous allez en trouver 34 orthographiés en majuscule. AX étant la représentation anglaise en phonème de la lettre A.
Auparavant, vous aurez remarqué que si vous utilisez le phonème Æ, vous entendez le son A (comme dans sac), alors que si vous utilisez le phonème AX, vous entendez le son E (comme dans leu), pourquoi ne pas, avec votre éditeur de blocs favori, aller remplacer les 34 occurences de AX par 34 Æ, et ainsi de suite pour que les phonèmes anglais deviennent des phonèmes français?

Toujours sous Appleworks 3.0, vous allez trouver en plus des phonèmes individuels, des groupes de phonèmes comme dAHbAXlyUW qui correspondant à la prononciation de la lettre W en anglais.
Il ne reste plus qu'à remplacer cet ensemble par dAWblvlH pour obtenir un W prononcé à la française.
Puisque vous en êtes aux groupes de phonèmes, le dictionnaire des exceptions du programme ou de l'outil comporte une série de phonèmes comme jÆnyUWEHrlY pour january, ou encore klOWz pour close, qu'il ne vous reste qu'à modifier en jEWvylY pour janvier ou flHrm pour ferme.
Vous êtes évidemment limités à la taille de la chaîne d'origine que vous ne pouvez pas dépasser, mais au bout d'un petit moment, l'expérience aidant, vous francisez entièrement le programme ou l'outil.
Une fois tous les phonèmes modifiés, vous constaterez qu'il est difficile, sinon impossible de faire prononcer un U correct français, ainsi qu'un son R. De même, vous avez francisé les chiffres, mais les 16, 17, 18 et 19 vous poseront un problème dans le sens ou en anglais ils se disent sixteen, seventeen, eighteen et nineteen, ce qui donnera en français sixdix, septdix, huitdix et neufdix. Qu'à cela ne tienne, vous vous **fabriquez alors votre propre dictionnaire des exceptions.**

Disponible aussi bien dans *SMOOTHTALKER* qu'en programmation, le dictionnaire des exceptions permet de faire correspondre une série de phonèmes à un mot. Par exemple, vous désirez que la chaîne de caractères *MOTO* ne soit pas prononcée *MOTO* , mais *VRAOUUM*, il suffit d'appeler le dictionnaire et d'inclure le mot *MOTO* avec sa représentation en phonèmes vrÆAWmmmm.
Quand le texte ou le programme rencontrera *MOTO*, il prononcera *VRAOUUM*.
Pour les chiffres mentionnés ci-dessus, pour les noms propres ou pour toutes sortes d'animations, enrichissez le plus possible votre dictionnaire des exceptions.

La programmation de la parole de synthèse

Quelques mots sur la programmation de la parole de synthèse. Outre les outils de démarrage ou de fermeture de la parole (*SpeechStartup* ou *SpeechShutDown*), vous avez deux possi- ▶

bilités différentes pour faire parler le II GS. L'instruction *SAY* ou l'instruction *PARSE*.

L'instruction *SAY* se contente de récupérer une chaîne de caractères, de l'interpréter et de la prononcer. Les seuls réglages possibles sont le Genre, la Tonalité, la Hauteur, la Vitesse, et le Volume en utilisant avant l'instruction *SAY*, l'instruction *SetSayGlobals*(Genre,Tonalité,Hauteur,Vitesse,Volume) qui effectue les réglages voulus.

EXEMPLE EN C DE L'INSTRUCTION SAY

```
ParoleDirecte(Son,Voix)
int Son;
int Genre;
{
        SetSayGlobals(Genre,Tonalité,Hauteur,
        Vitesse,Volume);
        chaine[0]=(char) strlen(chaine+1);
        Say(chaine);
}       /* fin de la fonction parole directe */
```

L'instruction *PARSE* est beaucoup plus puissante car elle traduit une chaîne de caractères en phonèmes. Une fois les phonèmes obtenus, en appelant les instruction *MALESPEAK* ou *FEMALESPEAK*, vous faîtes dire exactement ce que vous désirez avec la modulation désirée dans la voix.

Par exemple, le prénom *CAMILLE* n'est pas dit d'une façon uniforme, tout sur le même ton. Vous élevez le ton pour le début du mot, puis vous baissez le ton pour la fin du mot. En faisant prononcer des phonèmes, vous pourrez faire la même chose, vous écrirez 5kÆmyyyyy3AX, ce qui donnera un effet d'affaiblissement de la prononciation de la fin du mot, chose impossible à obtenir avec l'instruction *SAY*.

EXEMPLE DES INSTRUCTIONS PARSE ET MALESPEAK EN C

```
TextenPhoneme(Son,Voix,PhVol,PhVit,PhHau)
int Son;
int Genre;
{       /* traduit texte normal en phonèmes */
        Parse(chaine,Phonetique,1);

        /* appel à la fonction pour prononcer
        les phonèmes */
        ParlePhoneme(Son,Genre,PhVol,PhVit,
        PhHau,Phonetique);
}       /* fin de la fonction TextenPhoneme() */

ParlePhoneme(Son,Genre,PhVol,PhVit,
PhHau,Phonetique)
int Son, Genre, PhVol, PhVit, PhHau;
char Phonetique[];
{
        MaleSpeak(PhVol,PhVit,PhHau,Phonetique);
        /* Volume, Vitesse, Hauteur, Adr chaîne */
}       /* fin fonction ParlePhoneme() */
```

Vous aurez remarqué au passage que la parole de synthèse permet aussi de changer le genre de la voix, genre homme ou genre femme, soit en passant un paramètre à 0 ou 1 à *SAY*, soit en utilisant *PARSE* puis *MALESPEAK* ou *FEMALESPEAK*.

Pour plus de détails sur toutes ces fonctions et les autres fonctions de *TML SPEECH TOOLKIT*, se reporter à la documentation et aux exemples.

Après avoir fait un petit tour de la parole de synthèse, il est temps de se consacrer à la parole digitalisée.

Parole digitalisée

Comme le mot l'indique, il faut digitaliser le son pour obtenir de la parole digitalisée. Comme pour la parole de synthèse, cette méthode comporte des avantages et des inconvénients.

L'avantage est bien sûr la qualité du son obtenu, l'inconvénient est la taille du fichier à utiliser et le manque d'interactivité avec l'utilisateur.

Comment obtenir non seulement de la parole, mais plus généralement un son digitalisé? A partir d'une carte de digitalisation de son. Actuellement, et suivant vos moyens, vous avez 3 possibilités.

Comment digitaliser

La solution la moins chère est d'acheter le programme *HYPERSTUDIO*, car non seulement vous avez le programme, mais en plus une carte de digitalisation et un micro. En revanche, impossible de sortir en stéréo sur votre chaîne.

Un tout petit peu plus cher, mais guère plus, la *carte Sonic Blaster* qui comporte une entrée stéréo et une sortie stéréo par l'intermédiaire de deux prises type jack 3.5. Le programme livré avec permet de digitaliser à différentes fréquences, depuis 5.082 HZ jusqu'à 30.368 HZ.

Plus vous utilisez une fréquence basse, moins importante sera la taille du fichier pour une durée identique, mais, revers de la médaille, moins le son sera de bonne qualité. Vous voulez digitaliser le bruit d'un coup de pistolet, 8.000 HZ pourra aller, vous voulez digitaliser une voix, le minimum est 10.000 HZ, quant à de la musique, en dessous de 15.000 HZ, la qualité ne sera pas au rendez-vous. Un bon rendu des sons ▶

s'obtient avec une fréquence d'environ 21.000 HZ, mais attention, un fichier devient vite énorme…
Autres possibilités du programme, jouer avec les sons. Vous pouvez les copier/couper/coller, faire prononcer à l'envers, mettre de l'écho, faire débuter le son de zéro jusqu'au volume normal (*fade up*), faire arrêter le son en douceur (*fade down*), faire accélérer, ralentir la vitesse, etc, bref de quoi réaliser d'excellents effets sonores et amuser les petits et les grands.

Dernière solution pour digitaliser les sons, la *carte Audio Animator*, qui diffère essentiellement de la Sonic Blaster par sa possibilité de gérer les entrées/sorties Midi et donc à raccorder au II GS n'importe quel instrument Midi. Je ne m'étendrais pas sur cette partie musicale, laissant aux charpentiers le soin de le faire.
L'*Audio Animator* est équipée d'une petite boîte qui se place à côté du II GS et sur laquelle viennent se raccorder toutes les entrées/sorties. Que ce soient les entrées/sorties Midi avec des prises *DIN*, les entrées/sorties son stéréo avec des prises *RCA* (deux en entrée et deux en sortie), l'entrée Micro et la sortie Casque.
Sur le dessus de la boîte, cinq curseurs règlent le volume de sortie (canal droit et canal gauche), le volume d'entrée (canal droit et canal gauche) et le volume d'entrée du micro.
Le programme pour digitaliser ressemble comme deux gouttes d'eau à celui de la Sonic Blaster, à part deux ou trois modifications, comme les fréquences de digitalisation qui vont de 5.262 HZ à 37.000 HZ.

Une fois que vous avez digitalisé votre son, qu'il vous plaît comme il est, et que vous voulez le sauvegarder, plusieurs possibilités de sauvegarde s'offrent à vous. En type Æ Format, 2:1 Compressed, Raw Data, AIFF Format ou 2.67:1 Compressed.
Libre à vous de sauvegarder votre fichier dans le format que vous utiliserez par la suite, de toutes façons, si une fois le fichier sauvegardé dans un format vous décidez de le sauvegarder dans un autre format, il suffit de le rappeler, puis de le resauvegarder avec un format différent.
[NDLR: attention cependant, la compression de fichiers sons n'est pas une compression "à la Shrinkit", en ce sens que le fichier une fois décompressé n'est plus identique au fichier original, les sons sont légèrement "lissés". Décompresser et recompresser plusieurs fois le même fichier son dégrade progressivement sa qualité, de la même façon que les copies de copies de copies de cassette audio dégradent le son d'origine. Il est conseillé de conserver un exemplaire du son en format non-compressé.]

De la même manière, une façon de franciser des programmes américains utilisant les sons digitalisés (*Chessmaster 2100* ou la série des *Talking… Talking colors, Talking dinosaurs*, etc) est de charger un son du programme, de voir à quelle fréquence et sous quelle forme il a été créé, puis de digitaliser le même son en français avec les mêmes caractéristiques pour remplacer le son d'origine.

Comment utiliser les sons digitalisés

Plusieurs possibilités. Celle évoquée plus haut pour changer les sons d'un programme, simplement jouer avec les sons pour se fabriquer par exemple son message sur le répondeur téléphonique, ou utiliser les sons dans ses propres programmes, ou dans les piles HyperStudio, ou les piles HyperCard.

Utiliser les sons en programmation

Tous les langages permettent de récupérer les sons digitalisés. L'interface avec la Toolbox s'utilise de la même manière, voici en C comment récupérer un son :

EXEMPLE DE RECUPERATION D'UN SON DIGITALISE EN C

```
/* Paramètres pour struct fichier son */
SoundParamBlock sonParms1;
son(BufSon,TailSon,tempor,volume)
Pointer BufSon;     /* Pointeur sur l'adresse
                       du son en mémoire */
Long TailSon;       /* Taille du fichier son */
int tempor;         /* Fréquence pour restituer
                       le son, 400 pour 21803HZ*/
int volume;         /* Volume du son */
{
        sonParms1.waveStart = BufSon;
        sonParms1.waveSize = TailSon/256;
        sonParms1.freqOffset = tempor;
        sonParms1.docBuffer = 0;
        sonParms1.bufferSize = 0;
        sonParms1.nextWavePtr = 0L;
        sonParms1.volSetting = volume;

         /* on arrête le son en cours */
        FFStopSound(0x7fff);
         /* on joue le son demandé */
        FFStartSound(0x0001,&sonParms1);
}       /* fin de la fonction son() */
```

Il suffit ensuite d'appeler la fonction *son()* avec les bons paramètres pour jouer le son digitalisé.

▶

Dans une pile HyperStudio

Tout d'abord, aller changer le type du fichier son digitalisé pour qu'il soit reconnu par *HyperStudio*. C'est très simple, il suffit de lancer *SoundShop*, livré avec *HyperStudio*, de charger le son, puis de le sauvegarder.
Ensuite, quand vous crééz un bouton dans une carte *HyperStudio*, vous cochez devant *"play a sound"* (jouer un son). Le programme vous demande de cliquer le son, puis une fois le son trouvé vous propose de l'écouter et enfin, vous propose de l'incorporer à la pile ou de le laisser sur disque. Dans le premier cas, la pile sera plus importante en taille, mais le son sera joué quasi instantanément, dans le second, un même son peut servir à plusieurs endroits et bien qu'il faille un petit temps de réponse, la taille de la pile restera raisonnable.
Avec *HyperStudio*, contrairement à la programmation et à HyperCard, pas besoin de tenir compte de la fréquence de digitalisation, le son est restitué à la bonne fréquence. En contrepartie, il n'est pas possible de jouer avec le son en le faisant aller plus ou moins vite, ou en n'en faisant jouer qu'une partie.

Dans une pile HyperCard

HyperCard gère les sons d'une manière tout à fait différente puisqu'il utilise les ressources. Cela supprime la question de la gestion de la mémoire, le Resource Manager s'en charge.
Première constatation, il faut transformer le fichier son en ressource. Qu'à cela ne tienne, on prend *GENESYS*, on transforme le son en ressource en l'incluant au passage dans la pile HyperCard, on lance la pile HyperCard, on crée un bouton associé au son, on écoute et BOING le II GS est planté!
Pas de panique, on recommence à zéro et on va y arriver, mais c'est un petit peu plus compliqué qu'en programmation ou qu'avec HyperStudio, en revanche le résultat en vaut la peine.
Genesys, éditeur de ressources, est incapable d'ouvrir une pile HyperCard qui ne contient pas de ressource. Il faut donc d'abord transférer une ressource son (ou autre) dans notre pile grâce à la commande *RESCOPY* d'HyperCard.
Deuxième constatation, en prenant un même son, puis en le sauvegardant sous des types différents (Æ Format, RAW Data ou HyperStudio), on s'aperçoit que le son digitalisé ne débute vraiment qu'après une série plus ou moins grande d'octets, cette série d'octets étant des indications se rapportant au fichier pour chaque type. HyperCard pour sa part, demande une série de 10 octets nuls avant le début du son, avec le 3ème octet représentant l'octet de poids fort de la taille du son.
Exemple, soit un son digitalisé d'une taille de 19520 Octets, sa taille est en Hexa de $4C40, l'en-tête du fichier son à incorporer dans HyperCard devra être :
00 00 4C 00 00 00 00 00 00 00

Comment inclure un son dans HyperCard

Voici comment modifier un fichier son pour l'inclure dans HyperCard. Plusieurs solutions s'offrent à vous, suivant les programmes ou les accessoires de bureau que vous possédez.

• Première solution, la plus simple, *CUSTOM REZ V1.5*. Ce freeware, écrit par Mlle Carole Fox (auteur entre autres du Jeu de l'oie et de Mémocartes) charge un fichier son et l'incorpore directement dans une pile HyperCard après avoir modifié l'en-tête du fichier. On ne peut pas faire plus simple, mais tout le monde n'a pas ce freeware.

• Deuxième solution, *MOULISONS*. Ce freeware de votre serviteur permet de charger un son type Æ Format, RAW Data ou HyperStudio et de le sauvegarder en modifiant l'en-tête du fichier. Il ne reste plus ensuite, avec *GENESYS*, qu'à charger la pile HyperCard, et importer le fichier son modifié par Moulisons pour l'incorporer en ressource son (type $8024) dans la pile. N'oubliez pas d'aller sur Attributes pour avoir le nom de ressource que vous voulez et de cocher le Niveau 2 de purge.

• Troisième solution, *DUMP*. Cet autre freeware, écrit par Michel Falempin, permet de charger en mémoire un fichier, puis de supprimer une série d'octets (les premiers concernant le type du fichier), puis de rajouter une série d'octets (les 10 octets demandés par HyperCard), et enfin de modifier la valeur des 10 octets rajoutés, tous à zéro, sauf le 3ème.

• Dernière solution, avec *PEEKIT*, *WRITEIT* et un éditeur de blocs. Solution un peu plus compliquée, sans être difficile, mais utile à connaître si vous n'avez pas sous la main les programmes pré-cités.
Prenez l'accessoire de bureau *PEEKIT*.
Chargez votre fichier son.
Sélectionnez uniquement la partie son proprement dite, sans les indications de type de fichier du début.
Copiez avec le menu *EDIT*, fermez *PEEKIT*.
Prenez l'accessoire de bureau *WRITEIT*.
Collez avec le menu *EDIT* ce que vous avez



copié précédemment.
Placez votre curseur en début de fichier et insérez 10 lettres, par exemple le chiffre 0.
Sauvegardez le fichier dans le dossier que vous voulez, mais surtout sous format Texte.
Fermez *WRITEIT* et lancez votre éditeur de blocs en sélectionnant avec l'éditeur le fichier sauvegardé sous *WRITEIT*.
Modifiez les 10 premiers octets en mettant des 00 à la place
Modifiez le 3ème octet en mettant la valeur hexa de l'octet de poids fort de la taille du fichier.
Ecrivez le nouveau bloc à la place de l'ancien, votre fichier est prêt à être incorporé dans une pile HyperCard avec Genesys.

Jouer un son dans HyperCard

Votre son, digitalisé, modifié pour l'en-tête, transformé en ressource et inclus dans votre pile HyperCard est maintenant prêt à être joué.
Grâce au langage HyperTalk, vous allez pouvoir jouer le son quand vous le désirez, que ce soit à l'ouverture d'une pile, d'une carte, en passant la souris sur un bouton, en cliquant, bref quand et où vous le voulez.
Contrairement à HyperStudio, le son avec HyperCard est géré comme une note de musique. Même si votre son dure trois minutes, pour HyperCard, ce sera une note de trois minutes. Avantage de cette technique, vous pourrez gérer votre son en le faisant jouer avec la hauteur de note que vous voudrez et dans l'octave que vous voudrez.
En pratique : vous avez inclus dans votre pile le son *BONJOUR*. Vous désirez qu'à l'ouverture de votre carte, on entende *BONJOUR*, cliquez sur *CARD INFO*, puis sur *SCRIPT*.
Une boîte de dialogue s'ouvre, dans laquelle vous allez taper :

```
on openCard
  play"bonjour"
end openCard
```

Cliquez sur OK, la prochaine fois que vous ouvrirez votre pile, vous entendrez *BONJOUR*. Seul léger problème, si HyperCard, comme vu précédemment, gère le son comme une note de musique, il prend par défaut une fréquence d'environ 21.000 HZ. Ainsi, si votre son a été digitalisé à cette fréquence, le son sera joué normalement, en revanche, si votre son a été digitalisé à 10.000 HZ, il sera joué deux fois trop vite. Il faut alors modifier la vitesse d'exécution en donnant la note et l'octave correspondant. Pour une vitesse de 10.000 HZ, il faut mettre G2, ce qui correspondant à un SOL 2ème octave. HyperCard reconnait les notes américaines (C = DO, D = RE, E = MI, F = FA, G = SOL, A = LA, B = SI).
Pour que le son soit joué avec la bonne vitesse, il faut utiliser l'instruction *TEMPO* qui détermine la vitesse d'exécution, il faut aussi associer à la note, la durée de la note (ronde, blanche etc...) ce qui donne, toujours pour *BONJOUR* :

```
on openCard
  play"bonjour" tempo 70 "g2w"
end openCard
```

La valeur 70 détermine la vitesse d'exécution de la note, en conséquence, si des grésillements se font entendre après le *BONJOUR*, ou si le temps de réponse de la commande suivante est trop long, la vitesse est trop longue, il faut plutôt mettre 80, voire 90 comme valeur. En revanche, si *BONJOUR* n'est prononcé que *BON*, par exemple, il faut rallonger la vitesse et mettre 60 ou 50.
La lettre W correspond à une ronde (W = ronde, H = blanche, Q = noire, E = croche, S = double croche, T = triple croche, X = quadruple croche). Dernière précision, il est aussi possible de mettre des altérations, bémols ou dièses (Gb2W pour une Ronde SOL Bémol 2ème octave).

Pour plus d'informations sur les possibilités (très étendues) du langage interprété HyperTalk, reportez vous à la documentation du programme, au livre sur HyperTalk GS édité par Addison-Wesley, ou aux guides Marabout sur HyperCard et HyperTalk.

A présent vous avez tous les éléments pour faire parler votre II GS, à vous de choisir entre la parole de synthèse et les sons digitalisés, entre l'utilisation des programmes existants ou la programmation. **En fin de compte, pourquoi ne pas tout essayer ?** ■

A propos des ressources sons

Il existe désormais un format officiel Apple pour les ressources sons-digitalisés ($8024), telles que les utilise par exemple HyperCard. Ce format est détaillé dans la *Note Technique Apple II GS N°76*, et repris dans le livre *HyperCard II GS Script Language Guide, The HyperTalk Language (Addison-Wesley)*, page 147.
Il y a bien 10 octets d'en-tête, mais ces 10 octets ne se limitent pas à la taille du fichier en pages. Il semble cependant qu'HyperCard se contente de la valeur évoquée par Jacques Liautard: ça marche en tout cas comme ça, voyez les piles sur le disque.

JYB

Deux éditeurs pour APW/Orca

Bernard Fournier

En début d'année sont apparus sur le marché deux produits concurrents qui faisaient cruellement défaut: des éditeurs de sources multi-langages et multi-fichiers pour APW/Orca.

Jusqu'à présent, l'éditeur texte d'APW/Orca ne permettait d'éditer qu'un seul fichier à la fois: désormais on dispose d'outils puissants permettant l'édition simultanée de plusieurs sources de toutes origines (Assembleur, C, Pascal, scripts, etc). De plus, des fonctions très performantes de recherche/remplacement et de couper/coller rendent ces éditeurs indispensables.

MaxEd

MaxEd 1.0 est un produit de 360 Micro Systems (éditeur de différents utilitaires APW/Orca).

Cet éditeur est certainement le plus performant, mais aussi le plus complexe. Il se présente sous la forme d'un utilitaire et d'un dossier à installer dans le dossier /UTILITIES de APW/Orca, ainsi que d'un outil (le tool 56) à mettre dans /SYSTEM/TOOLS de la disquette de démarrage. Toutes les fonctions très performantes de MaxEd résident dans l'emploi d'un outil spécifique (dont il devrait être possible d'utiliser les fonctionnalités à partir d'autres applications… mais cela est une autre histoire !).

Lorsqu'on lance MaxEd, on arrive dans un menu offrant trois possibilité: passer dans l'éditeur, configurer ce même éditeur, et enfin employer les utilitaires système.

L'éditeur de MaxEd

MaxEd permet le chargement en mémoire de 30 (!) fichiers sources simultanés accessibles par une commande de bascule (-Q). Chaque source peut être de toute origine: assembleur (y compris Merlin), C, Pascal (y compris TML), texte, link, script. Pour chaque type, une règle de tabulation spécifique est définie, permettant de conserver la mise en page lors d'éditions ultérieures dans l'éditeur spécifique.

L'éditeur MaxEd dispose de **fonctions de couper/coller** extrêmement riches: soit de manière interne (on déplace un bloc de texte à l'intérieur du même source), de manière locale (un bloc de texte est archivé dans un presse-papiers lié au source), de manière globale (un bloc de texte est archivé dans un presse-papiers disponible à partir de n'importe quel autre source), dans un presse-papiers Annulation (ce qui est effacé par -D est archivé dans ce presse-papiers et est récupérable par -Z), et enfin *via* le presse-papiers système (archivage dans un fichier sur disque). On peut aussi purger ces presse-papiers et les commuter: le global devenant local et réciproquement.

Comme pour tout éditeur qui se respecte, la **gestion des déplacements** est très souple: lettre par lettre, ligne par ligne, mot par mot, page par page et sur les tabulations; le tout bien entendu en avant/arrière et haut/bas; ainsi qu'un déplacement par fraction relative (-1 à 9) dans le style Appleworks.

On dispose également de **fonctions d'impression** très performantes: impression de la page écran, du fichier (en totalité ou partie sélectionnée) avec configuration du périphérique (imprimante Imagewriter, port série, port modem, réseau Appletalk, port disque...).

Une **aide en ligne** (-?) permet de visionner rapidement la totalité des commandes disponibles, et on peut définir un délai de **sauvegarde automatique** (de 30s à 20 mn) afin de ne pas perdre intempestivement le travail d'une nuit !

Fonctions spécifiques

Quelques fonctions spécifiques à cet éditeur facilitent grandement la vie du programmeur:

- **affichage des 256 caractères ASCII** avec possibilité de copier/coller tout caractère vers le document (on aura ainsi la possibilité de taper des codes ASCII supérieurs à 128 pour des affichages graphiques)
- **calculs 'à la volée'**: saisie de formules acceptant des données en hexadécimal, binaire, décimal ou ASCII (il manque l'octal pour les C-maniaques) et résultats affichés dans les 4 modes.
- définition de **macros sauvegardables**: les



macros sont associées à des séquences de touches programmables (par exemple option-caractère, option-⌘-caractère, option-shift-caractère). Les macros sont actives tant dans l'éditeur que dans les menus de configuration.
• **macros pré-définies** pour passer très rapidement d'une fonction à l'autre ou d'un fichier à l'autre.

La configuration de l'éditeur

Dans MaxEd, on peut:
• définir et archiver ses propres **règles de tabulation**;
• définir les **options d'impression** (lignes par page, taille des caractères, séquence d'initialisation, port imprimante);
• **définir ses propres commandes**: en standard est fourni un jeu de commandes APW (les commandes sont semblables à celles de l'éditeur d'APW) et un jeu standard (les commandes sont semblables à celles des éditeurs de texte classiques). Mais on peut aussi créer son propre jeu (par exemple définir ⌘-E pour Effacer au lieu de ⌘-D)
• **spécifier les fichiers à charger** à chaque lancement de MaxEd (ce qui évite d'avoir à le faire manuellement). Outre cette fonction Auto-load, on peut indiquer des fichiers à charger lors de la frappe de la commande de lancement de MaxEd, ou éditer tout fichier à notre convenance à tout moment à partir de l'éditeur.

Les utilitaires système

Pour le moment, MaxEd ne dispose que d'un utilitaire, le **MiniShell**. Cet utilitaire permet de disposer de toutes les fonctions du shell APW/Orca. Dans la documentation, il est spécifié que certains 'plantages' peuvent survenir lors de l'emploi de certaines commandes dans le mini-shell, et les auteurs attribuent ces inconvénients à APW/Orca. En attendant les nouveaux utilitaires de développement ByteWorks (disponibles dans les tous prochains jours) dont on peut espérer qu'ils n'auront plus ces bugs fâcheux, on peut aisément contourner le problème.
En effet, un des aspects fabuleux de MaxEd consiste **à garder présents en mémoire la totalité des fichiers même lorsqu'on lance une autre application**! On peut donc quitter MaxEd, lancer un utilitaire APW/Orca (ou toute application de votre choix) et lorsqu'on revient dans MaxEd, les 30 fichiers avec leurs presse-papiers sont encore là (du moins dans la mesure où le Memory Manager n'a pas été contraint de purger la mémoire pour cause d'encombrement massif: ce qui prouve une fois de plus que les prévoyants ayant équipé leur GS d'une carte RAM additionnelle de 4 mégas ne le regretteront pas).

Les inconvénients de MaxEd

Ils sont rares, mais on peut signaler:
• la non-conversion des codes ayant le bit 8 positionné (ce qui permettrait de récupérer les fichiers TML sans les I-inverse à la place des tabulations);
• l'absence de multi-fenêtrage dans l'éditeur;
• la modification du langage dans le tableau de bord: le système 5.04 vf permet en effet l'affichage des caractères étendus via l'option FRANCAIS, mais en quittant MaxEd cette option est remise sur USA.

Edit 16 v1.0

Edit 16 v1.0 est un produit de Simple Software System International (l'éditeur de Genesys).
Cet éditeur, sorti en même temps que MaxEd, est très performant, ayant privilégié la facilité d'emploi sur la richesse des commandes. Si on le compare à son concurrent, on note les points de différence suivants:

Les 'moins'

• édition simultanée de 9 fichiers seulement, sans conservation en mémoire lorsqu'on quitte l'éditeur;
• ne conserve pas les types des fichiers édités (par exemple le TML est analysé comme un simple fichier Texte);
• presse-papiers classique: moins riche que celui de MaxEd mais plus simple d'emploi diront certains. Affaire de goût;
• 9 macros définissables seulement: associées aux touches numériques. Choix trop limité.

Les 'plus'

• affichage et impression du presse-papiers;
• multi-fenêtrage: mais on ne peut avoir que deux fichiers différents dans deux fenêtres horizontales; on regrette de ne pouvoir avoir deux zones différentes du même fichier (comme Applewriter le faisait sur nos antiques //e) [les ergoteurs diront qu'il suffit de charger deux fois le fichier en mémoire et ainsi on a deux versions du même source dans les deux fenêtres; mais cette solution n'est guère satisfaisante, car source de confusion: quel est le source de travail et celui de référence?];
• les fonctions souris (déplacement et clic) sont redéfinissables au clavier; ▶

• possibilité de visionner un fichier sans le charger;
• configuration de la largeur d'édition (maximum 256 caractères) par défilement horizontal de l'écran;
• changement de jeu de tabulations (5 maxi) 'à la volée', alors que dans MaxEd il faut aller dans un utilitaire spécifique de chargement/sauvegarde/édition;
• conversion à la demande des caractères ayant le bit 8 à 1 en caractères 7 bits lors du chargement: ainsi les fichiers TML sont listables sans les I-inverses remplaçant les tabulations.
• lorsqu'on est positionné sur une accolade ou une parenthèse, une fonction spéciale permet de trouver la parenthèse ou accolade de fermeture/ouverture associée, au même niveau hiérarchique. Ceci permet de se placer très facilement en fin ou début de procédure/fonction/formule, et ce à n'importe quel niveau d'emboîtement.

Conclusion

Ces deux éditeurs sont de puissants outils dont on attendait la venue avec impatience. MaxEd est très complet, mais du coup un peu 'lourd' à utiliser (mais une fois qu'on l'a bien en main, on est sidéré par sa souplesse). Edit16, quant à lui, est nettement plus souple au premier contact, mais à l'usage certaines fonctions absentes font cruellement défaut.

Je conseillerais à ceux qui disposent de peu de mémoire ou n'emploient qu'un seul langage de programmation d'utiliser Edit16. Quant à ceux qui ont investi dans 4 mégas de RAM ou qui jouent aux polyglottes des compilateurs, il est tout naturel qu'il s'offrent MaxEd.

Ces deux produits sont vendus à des prix tout à fait raisonnables, et le seul vrai reproche que l'on peut leur faire est qu'il ait fallu attendre si longtemps pour en disposer !

Adresses:

MaxEd
360 Micro System
P.O. box 1192
Oviedo, FP 32765, USA

Edit16
Simple Software System International
4612 North Landing Drive NE
Marietta, GA 30066,USA

■

Chic, un nouvel impôt !

La Contribution Sociale Généralisée est une innovation qui rencontre de farouches adversaires: à savoir... les services payeurs des entreprises, qui doivent refaire tous leurs modèles de bulletins de paye.

Mais elle rencontre aussi de chauds partisans: à savoir... les développeurs, qui en profitent pour faire une mise à jour (enfin une mise à jour obligatoire!) de leurs logiciels de paye.

Parmi ceux-ci, le Dr Walusinski, dont le logiciel **MemoPaye**, diffusé par **Imagol**, et tournant sur Apple II GS, est désormais à jour.

MemoPaye fait la paye de 5 salariés maximum, tous en-dessous du plafond SS, même relevant de plusieurs conventions collectives différentes.

MemoPaye est particulièrement adapté à la rémunération des pimpantes secrétaires et des accortes infirmières du cabinet médical du Dr Walusinski et des cabinets médicaux français en général. Il s'avale tous ces calculs atroces genre “60% de la part patronale de la cotisation Urssaf, avec un abattement de 20% en-dessous du plafond SS dans la limite des 120 heures”, et en sort des bulletins de paie impeccables.

Ce logiciel est en Applesoft enrichi MemDos, en mode texte 80 colonnes, et cela convient parfaitement à une paye: la CSG serait la même en couleurs et avec des icônes.

Il est livré sur disquette 3,5, sous MemDos, et... c'est justement ça le problème:

• d'une part, je ne sais pas comment on peut transporter ses données depuis une disquette MemDos sur une disquette Prodos, et nos data risquent d'être bloquées.
• d'autre part, le MemDos en question est un peu antique. Par exemple, si vous ne mettez pas votre Tableau de Bord en 40 colonnes avant de booter la disquette, ça plante désagréablement.

Dîtes, Docteur, une version “vraiment GS”... ?

JYB

Imagol - 72 Bd Raspail - 75006 Paris

« Temps de travail »

Un Init à programmer soi-même

Patrick Desnoues

NDLR: Une innovation dans ce numéro de ToolBox-Mag. Pour vous aider à vous mettre à programmer le GS, nous publions un ***plan de programme****. A vous d'écrire, tester, débuguer le programme lui-même. Guidé par notre spécialiste Patrick Desnoues, vous allez écrire votre premier programme en assembleur GS !*
Notez que ce programme, qui écrit dans la Bram, est incompatible avec les programmes qui restaurent la Bram, comme celui de J.L. Darbon publié dans ToolBox-Mag 4. Cf ToolBox-Mag 4, pages 25-26.
Certains morceaux de source étant réutilisables pratiquement tels quels, nous avons mis le fichier texte correspondant à cet article dans le sous-catalogue /TEMPS.TRAVAIL de la disquette, pour vous éviter d'avoir à ressaisir. Attention cependant, le fichier est en Ascii étendu, avec des Tabs: vous aurez un peu de nettoyage à faire sous APW/Orca.

"Donne un poisson à l'affamé, il mangera une journée. Apprends-lui à pêcher, il mangera toute sa vie"*,* dit un proverbe chinois. Qu'est-ce qui nous a tous poussés à nous mettre à programmer, sinon la volonté que cette machine fasse exactement ce que nous voulons, nous? Voyant dans ToolBox Mag une suggestion de programme d'un lecteur, j'y ai vu une bonne occasion de répondre selon ce proverbe.

Le but de ce programme est de **comptabiliser le temps d'utilisation de votre ordinateur favori**. N'oubliez pas que le maximum syndical est de 39 heures par semaine...
Le programme ne devant pas poser de problème majeur, voici un plan d'organisation, avec des indications techniques: le tout devrait permettre à un utilisateur désirant commencer la programmation de sa machine d'y arriver sans trop de difficulté.
Type de programme : seul un init permet ce genre de chose (voyez ToolBox-Mag 4, pages 5 à 13). Le langage utilisé ici sera l'assembleur APW ou ORCA.

Technique d'un init

```
MainProgram START

        Php         Sauvegarde le registre P
        Phb         Sauvegarde le registre B
        Phk         Le Registre B = Registre K
        Plb
        Long m,i    Registre en mode 16 Bits
        Sta MyId
```

Vérification qu'on est bien sous GS/OS 5.04: voir le source de JYB sur la disquette ToolBox Mag 4. C'est indispensable, puisqu'on utilise l'appel ConvSecondes décrit dans ToolBox Mag 3.

```
        ......
Programme
        ......
        Rtl
MyId  ds 2
        END
```

Principe du programme

- Sauvegarder dans un fichier sur disque, la date et l'heure où l'on allume le GS.
- Initialiser dans la Ram batterie un compteur qui va s'incrémenter tout le temps où le GS est allumé.
- Installer une tâche de fond qui va incrémenter cette valeur.
- Quand on rallume le GS, on va lire la valeur, on lui additionne la date de départ, et on va sauvegarder la date de fin de la dernière utilisation.

1) Si l'on vient d'allumer son GS :
Pour savoir si le GS vient d'être allumé, lire une valeur en mémoire : si elle est à zéro, on vient ▶

d'allumer. Changer cette valeur, pour pouvoir ensuite la tester, en cas de reboot. Elle ne doit pas être modifiée par un AutoTest... (ex : $01/03FF).

Lecture du fichier de données :

- *le fichier existe:*
  Lecture de la valeur de la ram batterie.
  Lecture de la dernière date enregistrée.
  Calcul de la date de fin d'utilisation.
  Sauvegarde de cette date.

- *le fichier n'existe pas:*
  Création du fichier (_CreateGS CRparms)
  Ecrire : 'Date de mise en marche / Extinction / Durée'

  Lecture de la date du jour
  Sauvegarde de cette date

  Initialisation du compteur
  Mise en place de la routine de fond

2) On vient de rebooter.

Réinstallation de la tâche de fond sans initialiser le compteur de la ram batterie.

Type de fichier de données :

Prendre un fichier de type texte que n'importe quel traitement de texte pourra consulter.
Positionner ce fichier sur le volume de boot dans le dossier */System/System.Setup/

Ecrire la date de début : Jour / Mois / Année - Heure
Mettre une tabulation
Ecrire la date de fin: Jour / Mois / Année - Heure
Mettre une tabulation
Ecrire le nombre de minutes utilisées.
Mettre un retour chariot (return)

date 1 (20 caract) Tab (1) date 2 (20 caract) Tab (1) nombre de minutes (5) CR (1)
soit 48 caractères par ligne.

Comment lire le fichier :

```
_OpenGS OPParms
Bcs Erreur

Lda Op_RefNum
Sta Cl_NumRef
Sta Re_NumRef

_NewHandle (Op_Eof+48)

_ReadGS ReParms
Bcs Erreur

_CloseGS ClParms
```

Lecture de la date :

Commencer par la fin du fichier en décrémentant.
Rechercher le caractère $0D (return)
Incrémenter la position de 1 : Date du jour
Convertir cette date en secondes (_ConvSecond décrit dans la disquette ToolBox-Mag n°3):

```
LongResult
Pushword #1     Format Hex du MiscToolSet
                -> Secondes
Pushlong #0
Pushlong #DateDuJour
_ConvSecond
PullLong Date1Secondes
```

Lecture de la valeur de la BRam:

```
    WordResult
    Pushword #$FA
    _ReadBParam
;   Pla

    WordResult
    Pushword #$FB
    _ReadBParam
    Pla
    Xba
    Adc 3,s
    Sta NbMinutes

    Plx
```

On multiplie le nombre de minutes par 60 pour avoir des secondes:

```
LongResult
Pushword NbMinutes
Pushword #60
_Multiply
PullLong NbSecondes
```

On additionne le nombre de secondes écoulées à la date de départ en secondes:

```
Add.L    Date1Secondes, NbSecondes,
         Date2Secondes
```

On convertit cette nouvelle date en secondes :

```
LongResult
Pushword #0    Secondes -> Format Hex du
               MiscToolSet
Pushlong Date2Secondes
Pushlong #DateDuJour+21
_ConvSecond
PullLong
```

Lecture de la date du jour :

```
Pushlong #DateDuJour+48
_ReadAsciiTime
```

Sauvegarde du fichier :

```
_OpenGS OPparms
_WriteGS WRparms
```

▶

```
    _CloseGS CLparms

; Paramètres OpenFile

OPParms          dc.w 15
Op_RefNum        ds 2
Op_Pathname      dc.L NameToLoad
Op_Req_Access    dc.w 0
Op_ResourceNum   dc.w 0        ; 0 : data / 1 : Fork
Op_access        ds 2
Op_Filetype      ds 2
Op_AuxType       ds 4
Op_StorageType   ds 2
Op_CreateTd      ds 8
Op_ModifyTd      ds 8
Op_OptionList    ds 4
Op_Eof           ds 4
Op_BlockUsed     ds 4
Op_ResourceEof   ds 4
Op_ResourceBlo   ds 4

NameToLoad  GsString '*/System/System.Setup/?'

; Paramètres ReadFile

ReParms          dc.w 4
Re_NumRef        ds 2
Re_Buffer        ds 4
Re_Eof           ds 4
Re_OctTrans      ds 4

; Paramètres WriteFile

WrParms          dc.w 4
Wr_NumRef        ds 2
Wr_Buffer        ds 4
Wr_Eof           ds 4
Wr_OctWrite      ds 4

; Paramètres CloseFile

ClParms          dc.w 1
Cl_NumRef        ds 2

; Paramètres Create

CrParms          dc.w 5
Cr_Pathname      dc.L 0
Cr_Access        dc.w AllAccess
Cr_Type          ds 2
Cr_Auxtype       ds 4
Cr_Storage       dc.w 1
```

Installation de la tâche de fond:

Page 14-50 volume 1 ToolBox Référence

```
    Pushlong #SaveBRam
    _SetHeartBeat
```

Toutes les minutes on incrémente un compteur sur 2 octets qui contiendra le nombre de minutes écoulées (en hexadécimal) depuis l'allumage du GS.

```
SaveBRam  Anop
    dc.L 0
    dc.w 60*60          Toutes les minutes
    dc.w $A55A
    dc.L 0
```

Si on arrive ici, c'est que la minute est écoulée et que le compteur est à zéro. On le remet donc pour le prochain passage:

```
    Lda #60*60
    Sta SaveBRam+4

    WordResult
    Pushword #$FA
    _ReadBParam
    Pla
    Ina
    Beq NotZero

    Pha
    Pushword #$FA
    _WriteBParam
    Rtl

NotZero  Anop
    WordResult
    Pushword #$FB
    _ReadBParam
    Pla
    Ina
    Pha
    Pushword #$FB
    _WriteBParam
    Pushword #0
    Pushword #$FA
    _WriteBParam
    Rtl
```

Si l'on veut initialiser le compteur à zéro à l'allumage du GS :

```
    Pushword #0
    Pushword #$FB
    _WriteBParam
    Pushword #0
    Pushword #$FA
    _WriteBParam
```

En résumé

Il est nécessaire de posséder un langage de programmation (APW / Orca / Merlin), les 3 volumes du ToolBox Reference (Addison-Wesley), une documentation GS/OS (ex : Exploring Apple GS/OS, par Gary Little), d'être abonné à ToolBox-Mag et d'avoir un peu de temps libre.

En cas de difficulté, on peut me contacter par l'intermédiaire de ToolBox-Mag. Bon courage à tous !

Made in France: la pénurie?

Bernard Fournier et J.Y. Bourdin

Première Partie: l'enquête de Bernard Fournier

Comment? Qu'est-ce? Qu'arrive-t-il? Pas de nouveautés dans cette rubrique ce mois-ci? Est-ce à dire que les programmeurs ont fait fortune avec leurs sharewares et sont partis sous les tropiques dépenser leurs magots? Il fallait en avoir le coeur net, et nos plus fins limiers ont sillonné inlassablement les chemins de notre beau pays afin de se rendre compte sur place.
Tout d'abord, direction Dijon. Dans une rue paisible, d'étranges bruits s'échappent par le vasistas d'un garage. Glissant un oeil par l'ouverture, notre enquêteur vit un groupe de joyeux drilles en effervescence autour d'un clavier et il entendit des bruits de canonnade. Bizarre, bizarre....
Poursuivant son chemin, il s'arrête aux portes de Lyon traquant les programmeurs fous. Dans les lointaines banlieues, aux marches du Dauphiné, il rencontre un gentil monsieur fort télégénique qui lui montre subrepticement une merveille de machine à écrire… à moins que ce ne fût un GS…
De plus en plus intrigué, et sur la foi d'informations dont on taira la source, notre Rouletabille pousse son voyage jusqu'au bord de mer. Là, il se heurte à une porte close. Par un interstice de persienne il distingue un homme s'affairant au milieu de milliers de pages de listings. Un seul coup d'oeil suffit à notre reporter pour distinguer très nettement la nature de ces pages: il s'agit du désassemblage d'une énorme application.
Cette fois, notre limier est carrément perplexe. Que ce passe-t-il dans ces chaumières? Surtout que la rumeur informatique lui colporte un arrivage important de joysticks dans une autre région de France... la même rumeur prétendant qu'ils sont destinés à une seule et même personne, enfermée dans sa chambre depuis 3 mois.

Bon sûr, mais c'est bien sang! Tous ces évènements ont un lien! Si la rubrique Made In France est vide ce mois-ci, et si les programmeurs fous sont enfermés chez eux en ces belles journées printanières, c'est qu'ils mettent la dernière touche à leurs créations! Donc pas de panique: l'été va être chaud!

Note: tous les événements et personnages décrits dans cet article existent réellement et toute ressemblance avec des programmeurs GS connus est volontaire. Concluez vous même…

Bernard Fournier

Deuxième Partie: les déductions de J.Y. Bourdin

Pour préciser les allusions de Bernard, je dirais que **la pénurie actuelle de programmes français nouveaux en freeware ou shareware est un excellent signe**.
Il semble bien, en effet, que les programmeurs GS français, entre les diverses formules **de publication et de commercialisation** existant en ce moment en France, aient enfin trouvé le moyen de **diffuser commercialement de façon normale leur travail**.
Quand *GS Plus* publie aux USA des publicités pour Photonix II, qu'*A2 Central* envoie sur toute la planète un catalogue vendant Space Shark et les autres, qu'*InCider/A+* s'y met à son tour, c'est que quelque chose est en train de changer dans le bon sens. La reconquête de l'Amérique serait-elle en cours?

Vrai et faux shareware

ToolBox-Mag lui-même, en publiant tous les bons programmes qu'on lui envoie, alors qu'auparavant il n'existait guère d'autre issue que le shareware ou le freeware, fait sans doute baisser la production de ce **"faux" shareware** qui ne prenait ce canal qu'à défaut de débouché commercial normal.
Le shareware se rétrécit donc peu-à-peu, en France, à ce qui est **sa vraie fonction: celle du "galop d'essai"** par lequel un auteur fait connaître ce dont il est capable.
Mais quand les galops d'essai sont terminés, on commence les vraies courses. Les programmeurs français du GS travaillent: mais il est fort probable que "Made in France" doive désormais étendre son domaine aux logiciels commerciaux.

▶

Dépêchez-vous

La preuve: je viens de recevoir un nouveau shareware, la **version 2.2** de l'indispensable **TransProg** de **François Uhrich** (voir les éditions précédentes de "Made in France"). Corrections de quelques défauts (TransProg ne se déclenche plus en haut de l'écran quand on a enlevé la barre des menus dans un programme de Paint, par exemple), meilleure gestion des menus, mais surtout de la mémoire (ressources et segments dynamiques permettent de ne prendre que 20k en mémoire sur 50), ajout d'une option redémarrer au menu éteindre.

Dans son état actuel, TransProg est un logiciel pratiquement "bug-free", de niveau largement commercial, supérieur à On Cue sur Mac (on peut trier la liste de TransProgMenu). En conséquence... **c'est la dernière version en shareware!**

La version 3.0 de TransProg est en cours de développement, mais nous dit François Uhrich, elle « ne sera plus diffusée sous la forme d'un shareware, mais d'un produit commercial à part entière », par BrainStorm Software, des conditions particulières étant alors accordées aux actuels acquéreurs. *Après Fontasm, Photonix, Bouncing Bluster, c'est le tour de TransProg. Il y a bien là un destin.*

Sur la même disquette, deux autres sharewares à essayer avant de les acheter.

Pour l'un, le Fonte/DA Installateur 2.0 du même François Uhrich dont vous savez déjà que c'est un indispensable, je ne parierai pas une carte II Plus pour Mac à bas prix sur sa durée de vie en shareware, étant donné qu'il a les mêmes qualités que TransProg. Si vous ne l'avez pas déjà, **dépêchez-vous**: comme TransProg, il risque d'être plus cher bientôt.

Le vrai shareware

Le second, en revanche, illustre bien la fonction classique du "vrai shareware", qui est plus de permettre de faire savoir de quoi son auteur est capable, que de vendre réellement un produit. Vendu 40F par BrainStorm Software, ce CDev de **Louis van Proosdij**, intitulé **TransWarp.SetUp**, permet de configurer la carte TransWarp GS et de forcer une configuration déterminée de celle-ci lors du prochain reboot.

S'il ne faisait que reprendre les fonctions qui sont celles du CDA de la Rom de la TransWarp, ou celles du CDev en domaine public de Doug MacIntyre appelé "TransWarp", il ne servirait à rien. Mais il fait un peu plus, essentiellement en permettant de régler la fonction Enable/Disable DataCache de la TransWarp.

[NB: l'auteur recommande de laisser en position Enable. Je ne le suivrai pas sur ce terrain: sur ma configuration, il m'arrive une fois sur deux de ne pas pouvoir booter jusqu'au bout en position Enable. Et dans ce cas, plus moyen de rectifier, les CDevs étant inaccessibles. Voyez ce qui marche le mieux sur votre configuration.

Dans la mienne, certains logiciels supportent le DataCache, d'autres non. Comme tout cela se résume à JSL BC/FF4C (DisableDataCache) et JSL BC/FF48 (EnableDataCache), je me débrouille avec des patches et de mini-fichiers lanceurs, un accessoire ne convient pas.]

Ce qui prouve bien qu'il ne s'agit pas d'un programme à visée réellement commerciale, c'est que l'auteur s'est donné le mal de réécrire en CDev un programme qui existe déjà sous forme de NDA: l'excellent accessoire en freeware "**TransWarpGS.Info**", de **E. et P. Lacquehay,** permet depuis Juillet 89 d'avoir accès aux réglages du DataCache, comme aux autres réglages de la TransWarp.

J'ai d'ailleurs du mal à voir ce qu'apporte le fait de présenter les mêmes fonctionnalités en CDev plutôt qu'en NDA: surtout qu'une des différences principales entre les deux (un CDev réside sur disque, et un NDA en mémoire) est désormais disparue, puisque précisément, avec le Fonte/DA Installateur de F.Uhrich qui est sur la même disquette, **les accessoires aussi résident sur disque**! Ceci dit, si vous n'avez pas déjà "TransWarpGS.Info", "TransWarp.Setup" est également un bon choix.

Un programme qui marche, mais qui est réservé aux possesseurs d'un matériel déterminé et "doublonne" un programme gratuit déjà existant, ne se donne pas toutes les chances au plan commercial: manifestement, l'auteur a surtout voulu prouver qu'il sait faire un CDev, ce qui correspond bien à l'esprit "galop d'essai" du shareware.

Après ce galop d'essai réussi, je parie que son auteur ne mettra pas longtemps, lui non plus, à entrer dans la course!

J.Y. Bourdin

E. et P. Lacquehay
Pack Evolution,
112 Ave du Gal Leclerc
54000 Nancy

BrainStorm Software
7 allée Murat, Bois Impérial,
54630 RichardMénil.

■

Un logiciel ToolBox-Mag :

ResDoctor

par Jean Destelle

[NDLR. Nouveauté dans ce numéro: un logiciel complet, avec sa documentation mais sans le source, ***ResDoctor****. Regardez la longueur du code-objet: il était bien impossible de mettre le source.* ***ResDoctor, à lui seul, vaut à notre avis le prix d'un abonnement à ToolBox-Mag!***
Qu'est-ce que ResDoctor? Eh bien, c'est le remplaçant du couple infernal Rez/Derez. Tout ce que vous ne pouvez pas faire avec Genesys, vous pourrez désormais le faire avec ResDoctor. ***ResDoctor est un compilateur-décompilateur de ressources complet.***
A sa façon, ResDoctor est même un ***éditeur de ressources****: vous pouvez lui faire remplacer Genesys pour éditer toutes les ressources, simplement il faudra le faire "à la main". En fait, si vous avez TML Pascal II et ResDoctor, vous avez un équipement complet pour programmer le GS. Vous pourrez écrire une application complète comme* ***Sélect****. La documentation complète est incluse dans les fichiers d'aide sur le disque.]*

ResDoctor est le SAMU des ressources, mais il n'est pas que cela: c'est aussi un éditeur de ressources très complet pour le GS. Le plus complet sans doute à ce jour pour le GS, capable de tout faire sans sortir de l'environnement DeskTop.

Conçu au départ pour combler les lacunes de TML Pascal II dans ce domaine, *ResDoctor* a été progressivement complété par la suite par plusieurs utilitaires différents qui en font un **analyseur (désassembleur) et un éditeur-compilateur de ressources** très complets.

ResDoctor permet de faire pratiquement tout ce dont vous pouvez avoir besoin à propos des ressources du GS. Entre autres:

• **contrôler, ressource par ressource, tous les éléments des ressources** appartenant aux types standard connues à ce jour.
ResDoctor reconnaît de nombreuses ressources standard récemment définies, comme celles des formats de TextEdit, encore inconnues de beaucoup.
L'accès direct à chaque ressource est très pratique. Chaque ressource analysée comporte tous les renseignements utiles. Non seulement ses propres paramètres, mais, en clair, les titres et chaînes de caractères auxquels elle se réfère.

• **éditer, recopier, modifier, supprimer, créer individuellement les ressources**; cela qu'elles soient de type standard ou non.

• transformer les ressources de tout fichier de type étendu en un **listing désassemblé** très clair: c'est sa **fonction analyseur**.

• **imprimer des analyses** soit par ressource, soit par type, soit par famille de ressources (celles qui sont reliées par une interdépendance), soit pour un fichier entier.

• **visualiser les fenêtres** avec tous leurs contrôles, afin d'en corriger l'édition immédiatement, ainsi que les fenêtres d'alerte, avec leur présentation définitive.

ResDoctor comporte **deux éditeurs**: un **éditeur simple** fonctionnant ressource par ressource, très rapide d'emploi, incorporant son propre compilateur, et un **Grand éditeur**, destiné d'abord à transformer tout fichier ressource existant en un fichier source complet immédiatement imprimable, puis à éditer et à compiler ce fichier source, avec toutes les possibilités habituelles de TextEdit: édition pleine page, utilisation du copier-coller, etc.

On a particulièrement soigné la **clarté des sources et analyses** produits par *ResDoctor*: il est facile d'y entreprendre toute vérification utile.
Tous les noms de paramètres de la ToolBox sont respectés. La disposition adoptée respecte les habitudes des programmeurs de tous langages. C'est une application en mode "desktop", dont l'emploi est tout-à-fait intuitif.

Le logiciel comporte aussi plusieurs moyens différents de **recopie de fichiers comportant des ressources**, permettant de **transférer des "res-forks" complètes ou des ressources indivi-** ▶

duelles d'un fichier à un autre.

Il permet également de **transformer en ressource la totalité de la data-fork de tout fichier** (ou bien, ce qui revient au même, un fichier entier de l'ancien type). On transforme par exemple tout fichier texte en une ressource-texte en un tournemain.

La programmation du GS, quel que soit le langage utilisé, fait maintenant forcément appel aux ressources, qui représentent une très forte proportion du programme total. Souvent des erreurs, des accidents se produisent dans leur manipulation. ***ResDoctor*** permet, très rapidement, et avec la plus grande sécurité pour les fichiers et les programmes, de **retrouver l'erreur et de la corriger**.

ResDoctor reconnaît tous les types standard de ressources actuellement définis et permet d'en ajouter de nouveaux, qui pourront à leur tour, dans la plupart des cas, devenir analysables. ***ResDoctor*** est en effet une **application ouverte**, grâce à l'usage… des ressources, bien sûr!

Conçu pour réparer des erreurs, il a été testé de façon sévère, et plusieurs erreurs de principe dans la définition des ressources soit par TML Pascal II, soit par la documentation officielle ont été corrigées.

C'est un logiciel français, écrit en TML Pascal II par un programmeur de GS pour des programmeurs de GS. Bien que conçu plutôt au début pour le Pascal, il est absolument indépendant de tout langage de programmation.
Il privilégie avant tout la sécurité d'emploi. Expérimenté depuis plus de 8 mois, il s'est avéré un outil extrêmement sûr, auquel on peut confier ses fichiers sans crainte.

Enfin ***ResDoctor*** est une solution économique pour règler le problème des ressources. Car il se suffit à lui-même et comporte toutes les fonctions de traitement de fichiers, de désassemblage, de listage, d'édition et de compilation.
Pour l'utilisateur de TML Pascal II, c'est la solution idéale. Il pourra créer rapidement les premières ressources de son application avec l'éditeur graphique de TML, qui est ultra-rapide, puis par la suite, faire tous les contrôles, les modifications et les adjonctions avec ***ResDoctor***, avec une extrême facilité.

La notice très complète, intégrée et accessible directement à chaque instant, explique non seulement le fonctionnement du logiciel avec beaucoup de détails et d'exemples, mais aussi comment les ressources sont utilisées par le GS.

Dans le sous-catalogue */Select* du disque contenant le sélecteur ***Sélect***, vous trouverez des sources de ressources, établis par ***ResDoctor***, qui sont utilisées dans ce programme. ■

Les types de ressources connus de ResDoctor

Le tableau ci-dessous indique les types standard actuellement documentés par Apple. Pour certains types, le format n'a pas encore été publié.
La mention "analysable" signifie que ResDoctor reconnaît ce type, et en produit automatiquement l'analyse au moment du désassemblage de la ressource.

Type	Nom de la ressource	ResDoctor
$8001	rIcon	analysable
$8002	rPicture	analysable
$8003	rControlList	analysable
$8004	rControlTemplate	analysable
$8005	rC1InputString	analysable
$8006	rPString	analysable
$8007	rStringList	analysable
$8008	rMenuBar	analysable
$8009	rMenu	analysable
$800A	rMenuItem	analysable
$800B	rTextForLETextBox2	analysable
$800C	rControlDefProc	analysable
$800D	rCtlColorTbl	analysable
$800E	rWindowparam	analysable
$800F	rWindParam2	non analysable
$8010	rWindColor	analysable
$8011	rTextBlock	analysable
$8012	rStyleBlock	analysable
$8013	rToolStartup	analysable
$8014	rResName	non analysable
$8015	rAlertString	analysable
$8016	rText	analysable
$8017	rCodeResource	non analysable
$8018	rCDEVCode	non analysable
$8019	rcDEVFlags	non analysable
$801A	rTwoRects	analysable
$801C	rListRef	analysable
$801D	rCString	analysable
$8023	rC1OuputString	analysable
$8024	rSoundSample	analysable
$8025	rTERuler	analysable
$8027	rCursor	analysable.

Le GS pour débutants

deuxième partie : anatomie du 5.04

Eric Weyland

Nous parlerons aujourd'hui de l'architecture du système d'exploitation de l'Apple II GS, GS/OS 5.0.4. Je fais référence ici à la dernière version du système français: la terminologie exacte est « disque système 5.04 avec GS/OS 3.03 ».

Un disque système fonctionnel

Un avertissement tout de suite: en raison d'un bug découvert à la dernière minute, et signalé dans le catalogue /DERNIERE.HEURE de la disquette ToolBox-Mag 4, la version effectivement distribuée du système 5.04 français ne correspond pas à la description qu'en donnait J.Y. Bourdin dans ToolBox-Mag 4. Le 5.04 français est livré en deux disquettes, et **la première disquette (appelée** ***/Disque.Systeme*****) n'est pas pleinement fonctionnelle** (elle ne contient pas *FRInit*, et a un *Sys.Resources* non francisé: il s'agit de permettre le fonctionnement de l'Installeur si vous n'avez qu'un seul lecteur de disquettes).

La première chose à faire est donc de **vous constituer un disque système fonctionnel**. Voici comment (si vous ne comprenez pas ce qui suit ou n'arrivez pas à le faire, lisez d'abord le passage intitulé "un environnement hiérarchisé" un peu plus loin dans cet article):

① Faîtes une copie de la disquette appelée */Disque.Systeme*. Renommez cette disquette */Mon.Systeme*. Passez dans le sous-catalogue */Mon.Systeme/System/System.Setup*. Détruisez le fichier appelé *Sys.Resources*. Si vous avez un GS Roms 01, détruisez les fichiers *TS3 et PS3*. Si vous avez un GS Roms 03, détruisez les fichiers *TS2 et PS2*. (Voyez mon premier article dans ToolBox-Mag 4 pour savoir quelles sont les Roms de votre GS).

② Copiez, depuis la disquette */Utilitaires*, les fichiers */Utilitaires/System/System.Setup/Sys.Resources.K* et */Utilitaires/System/System.Setup/Frinit504* dans le catalogue */Mon.Systeme/System/System.Setup*. Renommez ces fichiers une fois copiés en *Sys.Resources* et *Frinit*. Copiez également le fichier */Utilitaires/System/Tools/Tool029* dans */Mon.Systeme/System/Tools*. Copiez enfin le fichier */Utilitaires/System/Drivers/Appledisk.5.25* dans */Mon.Systeme/System/Drivers*.

Vous avez maintenant une disquette système fonctionnelle. C'est cette disquette que nous allons analyser. Mais nous allons aussi voir comment la modifier: de même qu'il fallait, avant de démarrer le GS, le configurer pour que ce soit votre GS, il faudra de même **configurer le système pour que ce soit votre système** (en fait, en détruisant ces fichiers TS2 ou TS3, nous avons déjà commencé à personnaliser le système...).

Au-delà du Finder

Prenez votre disquette */Mon.Systeme* et bootez la. Assurez-vous de la version de votre système en appuyant sur une touche avant l'apparition du thermomètre. Au bout de quelques secondes, vous devez être sous le Finder.

Le Finder ne fait pas réellement partie du système GS/OS: c'est une **application**, fournie par Apple, qui utilise le système. Cette application a deux fonctions: lancer des programmes, d'une part, gérer les disques d'autre part (c'est-à-dire effectuer un certain nombre de travaux utilitaires comme formater une disquette, faire des copies de fichiers, de disquettes, afficher le contenu d'un catalogue, etc).

Ce Finder existe aussi sur Macintosh, où il fonctionne de la même façon. Le Finder est un programme en mode "desktop". Cela signifie qu'il utilise menus déroulants, souris, fenêtres, clics et double clics de la souris, boîtes de dialogue, etc. Mon propos n'est pas de décrire le fonctionnement du Finder: la documentation Apple (celle du système 5.02, le Finder restant le même dans le 5.04) est là pour cela, et l'Apple II Service Team d'Apple France se fait un plaisir de répondre à vos questions à ce sujet.

Le Finder est une application bien ajustée aux réticences des débutants, non parce qu'il 

serait "facile" (les métaphores du Finder sur le "bureau", les "dossiers", etc, sont tout sauf évidentes), mais parce qu'il utilise au maximum des petites images (icônes) au lieu des mots, et que cela, particulièrement sur le GS avec la couleur, a un aspect séducteur et attirant.

Ces images peuvent symboliser à peu près n'importe quoi: sous-catalogues (représentés comme des "dossiers"), lecteurs de disquettes, disque dur, RAM Disques, programmes, fichiers, accessoires de bureau, etc. Travailler avec le Finder consiste à ouvrir et fermer des "fenêtres", ouvrir et fermer des "dossiers", faire glisser des images, faire des clics et des doubles-clics.

Mais les inconvénients du Finder sont rédhibitoires: d'une part, le Finder, qui date de l'époque du Macintosh 128k, est bien adapté à la manipulation des disquettes 3,5, mais pas à la gestion des gros volumes disques d'aujourd'hui. Il ne sait pas, par exemple, **présenter l'arborescence d'un volume** (voir ci-dessous), si bien que gérer un ou plusieurs disques durs avec le Finder est un travail de forçat.

D'autre part, les petites images du Finder constituent un masque, un filtre déformant, qui empêche de savoir ce qu'on fait, et fait commettre des erreurs.

Par exemple, sous l'opération apparemment unique qui consiste à faire glisser l'icône d'un fichier, le Finder **confond trois opérations fort différentes**: le déplacement d'un fichier dans l'arborescence d'un volume (qui se contente de déplacer les pointeurs du fichier dans le catalogue), la copie d'un fichier (d'un volume à un autre par exemple, qui fait un double du fichier), et (si jamais vous faîtes glisser l'image du fichier sur une partie de votre écran qui ne contient pas de "fenêtre" de disque), une troisième opération, interne au Finder, qui ne fait rien du tout sur le fichier!

Du coup, il est obligé d'introduire des distinctions artificielles et arbitraires (pour copier un fichier d'une branche à l'autre d'un même volume, il faut appuyer sur Option en faisant glisser l'image du fichier, alors que c'est strictement la même opération que la copie sur un autre volume): bref, au bout, on ne comprend plus rien.

Même si le Finder a fait la fortune des concessionnaires Apple en entretenant la confusion chez les utilisateurs, il faut absolument sortir de ses métaphores et de ses petites images, de son univers artificiel, pour **comprendre ce qui se passe réellement sur nos disques**. Je parlerai donc ici de catalogues et sous-catalogues, et non de dossiers, de fichiers à copier et à déplacer, et non d'icônes à faire glisser, etc.

Qu'on me comprenne bien: je ne vous demande pas de cesser d'utiliser le Finder. C'est une des applications du GS, qui a au moins un mérite, celui d'être livrée avec le système, pour le même prix. Ce que je refuse, c'est l'assimilation du Finder à un composant du système, alors que ce n'est qu'une application, dont l'usage est totalement facultatif. **Pour comprendre le système, il faut oublier l'interprétation déformante qu'en donne le Finder**.

Un environnement hiérarchisé

Le système GS/OS (contrairement par exemple au fichier System du Macintosh, qui inclut fontes et accessoires) n'est pas un unique fichier, c'est **un environnement hiérarchisé et ordonné**, avec des sous-catalogues spécialisés contenant chacun un certain type de modules spécifiques.

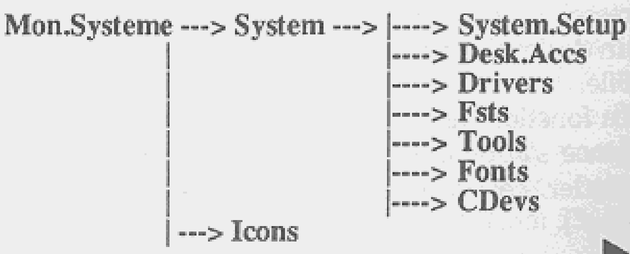
C'est qu'un système d'ordinateur moderne ne doit pas être conçu comme un programme, qui devrait être chargé en mémoire au lancement de la machine: c'est un **ensemble de ressources**, qui doivent être disponibles en permanence pour les applications, mais qui ne doivent être utilisées qu'en fonction des besoins et des diverses configurations.

La disquette que vous venez de booter s'appelle */Mon.Systeme*; elle contient un système d'exploitation de base permettant de faire le strict minimum. En clair, le système d'exploitation renferme des programmes et des données indispensables à l'Apple II GS pour faire fonctionner les programmes d'application (par exemple AppleWorks GS, HyperCard GS, ou le Finder).

Le système d'exploitation gère les divers périphériques: écran, clavier, lecteurs de disquettes, disque dur, disque virtuel, imprimante, etc. Comme cette configuration varie selon les utilisateurs, c'est à vous de **vous constituer votre propre système en le personnalisant pour la configuration que vous possédez**.

Le but de cet article est précisément de vous permettre de comprendre la fonction et la place de chacun des composants du système du GS, de façon que vous puissiez faire votre système, adapté à votre configuration.

Voici l'architecture générale de la disquette */Mon.Systeme*:



Vous avez sous les yeux une arborescence maintenant classique (cette architecture est commune à toutes les versions de GS/OS). Le Finder étant ce qu'il est, il n'est pas possible de visualiser cet arbre avec lui; c'est là une des grande limitation du Finder (y compris sur Macintosh). Dès que vous voudrez travailler avec votre GS, il faudra vous doter d'utilitaires sérieux; **ProSel 16** est à ce titre un excellent investissement.

Chaque branche de l'arborescence s'appelle un catalogue quand elle se rattache au tronc (le catalogue principal), et un sous-catalogue quand elle se rattache à une autre branche (le Finder traduit cela par les mots "dossier" ou "folder"). Il arrive également qu'on parle de sous-catalogue dès les premières branches, pour les distinguer du tronc: l'essentiel est de voir la distinction entre le tronc et les branches.

Ainsi, on dira qu'*/Icons* est un catalogue de */Mon.Systeme*, qui est le catalogue principal du volume, et que */Desk.Accs* est un sous-catalogue du catalogue */System* du volume */Mon.Systeme*. Un volume peut physiquement être une disquette, un disque dur, une partition d'un disque dur, un RAM Disque, etc.

Un catalogue ou un sous-catalogue peut lui-même contenir un ou plusieurs sous-catalogues. C'est le cas de */System*, catalogue du catalogue principal; ce catalogue */System* renferme plusieurs sous-catalogues: */System.Setup*, */Desk.Accs*, */Drivers*, */Fsts*, */Tools*, */Fonts* et */CDevs*.

Il est très facile de désigner un sous-catalogue particulier: il suffit de connaître et d'indiquer son "chemin d'accès". Pour désigner le sous-catalogue */Tools*, on écrira */Mon.Systeme/System/Tools*; chaque catalogue ou sous-catalogue est dans cette notation précédée du signe / (on peut aussi utiliser le signe :, mais Prodos 8 ne reconnaît que /).

Lorsque l'on veut sauvegarder un fichier de data, la commande de sauvegarde a toujours besoin de connaître le chemin d'accès du sous-catalogue où vous souhaitez sauvegarder votre fichier. Dans les applications en mode desktop, une boîte de dialogue vous demande de vous placer dans le sous-catalogue désiré (c'est à dire dans la branche de l'arborescence qui vous intéresse); ce système est toujours le même car c'est un des outils du GS qui s'en occupe: le Standard File.

La fonction de créer une branche dans l'arborescence s'appelle *créer un sous-catalogue* (pour le Finder, cela s'appelle "Créer un nouveau dossier").

Les constituants du système

Passons maintenant en revue les différents constituants d'un système de base (le disque */Mon.Systeme* 5.04 français). Pour cela nous procéderons sous-catalogue par sous-catalogue, car le système du GS (à la différence de celui du Macintosh, par exemple, où le catalogue /Systeme est un dépotoir universel) range ses composants chacun à sa place, de façon que nous puissions gérer proprement notre système.

Catalogue principal

Dans le catalogue principal, le fichier *Prodos* est responsable du chargement de tout le système GS/OS. Ce nom de *Prodos* ne doit pas vous faire croire qu'il s'agit ici du système d'exploitation Prodos 8 ou Prodos 16 (P16 est maintenant obsolète). En effet, le premier bloc des disques initialisés sous Prodos (responsable du chargement des disquettes et disques de l'Apple II GS) est écrit de manière à exécuter un fichier *Prodos* se trouvant dans le catalogue principal d'une disquette ou d'un disque. Si votre disque est initialisé, mais ne contient pas de fichier ayant pour nom *Prodos*, le message "Unable to load Prodos" s'affichera.

[Rappel: si un volume quelconque, /Ram5 par exemple, est catalogable et est muni d'un fichier *Prodos*, mais refuse de démarrer (booter), c'est parce que le premier bloc est vide: initialisez le disque avant de copier des fichiers dessus].

Le fichier *Prodos* peut faire a priori ce que l'on désire. Ainsi, sur les disquettes ToolBox-Mag, il y a un fichier *Prodos*. C'est bien entendu un pseudo-Prodos. Celui-ci se contente de faire un reboot sur la disquette, ce qui provoque le chargement de la superbe animation qui s'y trouve...

Le fichier *Prodos* des disquettes GS/OS est un chargeur (un loader) de tous les autres fichiers qui constituent le système, c'est lui qui lance tout le reste.

C'est aussi dans le catalogue principal que l'on trouve *Basic.System* et *Basic.Launcher*. *Basic.System* est le module qui permet d'utiliser des commandes Prodos 8 (Catalog, Open, Flush...) dans des programmes Basic Applesoft. Si vous n'utilisez pas de programmes Basic, vous pouvez effacer *Basic.System*.

Basic.Launcher permet de lancer des programmes Basic à partir du Finder. Là aussi, si vous n'utilisez pas de programmes en Basic AppleSoft, vous pouvez l'éliminer.

Catalogue /System

C'est dans le catalogue */System* que l'on va 

trouver un véritable *Prodos*. Il s'agit de Prodos 8, l'ancien système de l'Apple II 8 bits, que l'on trouve sous le nom de *P8*. Une manipulation classique consiste à faire booter une disquette en Prodos 8, afin de lancer *Basic.System*. Pour cela initialisez une disquette et copiez dans le catalogue principal (pour le moment, il ne peut pas y en avoir d'autre) les fichiers *P8* et *Basic.System*; renommez *P8* en *Prodos*. C'est tout: si vous bootez cette disquette, *Prodos* sera trouvé et exécuté; à son tour, celui-ci exécutera le premier programme X.SYSTEM (ici *Basic.SYSTEM*) qu'il trouvera. En tapant la commande "BYE", vous pourrez quitter le Basic; vous aurez alors sur l'écran ce qu'Apple a mis près de huit ans à mettre au point (ne ratez pas cette merveille: faîtes cette manipulation).
Si vous n'utilisez que des programmes 16 bits en mode natif, vous pouvez effacer le fichier *P8* de votre disquette système; le système ProDos 8 n'est alors pas nécessaire.

Dans ce catalogue */System*, vous trouverez surtout les principaux constituants du système d'exploitation GS/OS: *Start.GS.OS, GS.OS, Error.Msg, GS.OS.Dev* et *ExpressLoad*. Ces fichiers sont vitaux à la bonne marche du système; il est hors de question de les supprimer.
Le fichier ayant pour nom *Start* est ici le Finder. En effet, sur un volume GS/OS, après le chargement du système, c'est le programme *Start* du sous-catalogue */System* qui est exécuté. Ce programme *Start* peut être le Finder ou tout autre programme (par exemple ProSel 16 ou Select, le sélecteur de ce numéro). S'il n'y pas de fichier *Start*, le système recherche alors dans le catalogue principal du volume de boot le premier programme de type S16 se terminant par .SYS16, ou le premier programme de type SYS se terminant par .SYSTEM.

Changer de Start

A titre d'exercice, vous allez modifier votre disquette système pour que *Sélect* remplace le Finder: après tout on peut très bien s'en passer... Il est obligatoire de travailler sur une copie de votre système original: copiez la disquette */Mon.Systeme* et renommez la */TP.1*.
Nous allons maintenant faire un peu de place sur la disquette */TP.1*:
Dans le catalogue principal de */TP.1*, effacez les fichiers *Basic.System* et *Basic.Launcher*. Dans le catalogue */TP.1/System*, effacez le fichier *P8*, le sous-catalogue */Desk.Accs*, et le sous-catalogue */CDevs*, ainsi que les fichiers contenus dans ces sous-catalogues. Dans le catalogue */TP.1/System/System.Setup*, effacez le fichier *Cdev.Init*.
[Nous ne faisons ces suppressions que pour gagner de la place sur une disquette 800k: si vous faîtes ces manipulations sur votre disque dur, ne détruisez rien].
Nous avons maintenant la place pour copier *Sélect* sur la disquette */TP.1*. Copiez le fichier *SelectSys* de la disquette ToolBox-Mag sur la disquette */TP.1* dans le catalogue */System*. Renommez ensuite le fichier *Start* (le Finder) de la disquette */TP.1* en *Finder*, renommez *SelectSys* en *Start* et bootez la disquette */TP.1*.
Sélect est maintenant le programme de démarrage.
Les lecteurs courageux peuvent ranger le programme Finder à un autre endroit que le catalogue */System* où il n'a rien à faire. Il est bien entendu possible de placer le nom du Finder sur l'écran de votre nouveau sélecteur, afin de pouvoir lancer le Finder à partir de *Sélect*.
Pour utiliser le Finder, il est important de ne pas détruire le catalogue */Icons* du catalogue principal. Pensez aussi à appliquer au Finder le patch publié dans ToolBox-Mag N°2, page 41 (le Finder du 5.04 est le même que celui du 5.02), ou bien utilisez la version 2.2 du TransProg de F.Uhrich pour quitter proprement le Finder.
Enfin, le Finder a une fâcheuse tendance à polluer nos disques en posant des *Finder.Data* un peu partout. Ce n'est pas grave, mais c'est sale. La disquette ToolBox-Mag 4 vous a fourni un petit programme dépolluant d'Yvan Kœnig. Utilisez ce programme quand il est trop tard, mais songez surtout à "apprendre le caniveau" au Finder: cochez l'option "ne pas polluer les disques" dans le menu Préférences du Finder. Ceci fait, ne détruisez pas le fichier *Finder.Root* qui sera dans le catalogue principal de votre disque système (c'est lui qui dit au Finder de ne pas polluer). Je vous en remercie d'avance!

/System/System.Setup: les Inits

Examinons maintenant les différents sous-catalogues du catalogue */System* du volume */Mon.Systeme*.

Dans le sous-catalogue */System.Setup*, vous trouverez un certain nombre de modules de configuration que l'on appelle des Inits. Le système les utilise pour remédier à certains bugs de la rom. Ainsi, les fichiers *Tool.Setup*, *TS2* et *TS3* apportent ce type de modifications. Le fichier *TS2* n'est utile qu'aux personnes possédant un Apple II GS Rom 01; *TS3* à ceux d'entre vous qui possèdent un Apple II GS Rom 03: se- ▶

lon l'Apple II GS, Tool.setup va charger *TS2* ou *TS3*; vous pouvez donc effacer l'un des deux fichiers, et vous l'avez fait pour faire la disquette */Mon.Systeme*. Il est intéressant de constater que la taille du fichier *TS3* est nettement moins importante que celle de *TS2*; entre les deux, la Rom de l'Apple II GS a été mise à jour, et renferme donc moins de bugs...

Il existe deux sortes d'Inits: les fichiers d'initialisation temporaire (TIF) et les fichiers d'initialisation permanente (PIF): voyez notre dossier dans ToolBox-Mag 4 à ce sujet.

C'est aussi dans */System.Setup* que se trouvent *Sys.Resources* (type OS), et *FRInit*. C'est seulement si vous avez mis ces deux fichiers de la disquette */Utilitaires* à la place de ceux de la disquette */Disque.Systeme* que vous avez un véritable 5.04 français entièrement fonctionnel, avec la reconfiguration du clavier dite "configuration Kœnig" (voir ToolBox-Mag 3, page 16).

Attention, le fichier */Sys.Resources* est protégé par le système. Cela signifie qu'il ne peut pas être détruit sur votre volume de boot, car il est en permanence ouvert. Pour pouvoir le détruire et le remplacer, il faut lancer un système et aller faire cette manipulation en dehors du volume de boot. Il est d'ailleurs conseillé de faire tous les travaux utilitaires (optimisation, réorganisation des fichiers...) à partir d'un système qui n'est pas sur le volume sur lequel on veut faire ces travaux.

Dans le même sous-catalogue, le module *Resource.Mgr* est responsable de tout ce qui est ressources: il ne faut pas le supprimer.

Le module *CDev.Init* est utilisé en conjonction avec le tableau de bord graphique; si vous avez effacé ce NDA vous pouvez aussi supprimer ce module, ainsi que le sous-catalogue */System/CDevs* et les fichiers qu'il contient.

Panel.Setup est le module qui met en français le tableau de bord CDA (celui que nous avons étudié dans le numéro précédent). Selon la version de la rom de votre Apple II GS, *Panel.Setup* charge *PS2* ou *PS3*; là aussi, vous avez déjà dû supprimer l'un de ces deux fichiers pour faire la disquette */Mon.Systeme*. *Panel.Setup* peut être désactivé comme toutes les Inits et tous les accessoires: c'est indispensable si vous utilisez l'excellent SoftSwitch, de R. Wagner.

Il est très facile d'installer un fichier Init (TIF ou PIF): il suffit de le copier dans le sous-catalogue */System.Setup* de votre système. C'est par exemple le cas des inits TIF NICE.PATTERN, NICE.SCREEN et TOOLSTORAM que vous trouverez sur la disquette de ToolBox-Mag 4. A titre indicatif, il est utile de savoir que le programme TDM (The Desktop Manager) s'installe de cette façon, même si on y accède à partir du menu des accessoires de bureau (CDA). La dernière version (2.2) de l'indispensable TransProg (programme en shareware de François Uhrich) est aussi une Init à laquelle on accède à tout moment en déroulant un menu à droite sur la barre des menus des programmes en mode desktop.

Divers utilitaires permettent d'activer ou de désactiver une init. Le Finder permet de faire cela dans une des options de la commande "Lire les Informations". Lors du prochain chargement de GS/OS, selon ce que vous aurez choisi, l'init ou les inits seront ou ne seront pas exécutées.

/System/Desk.Accs et /System/Cdevs: les accessoires

Dans le sous-catalogue */Desk.Accs*, on trouve les accessoires de bureau. Il existe deux types d'accessoires de bureau: les CDA (Classic Desk Accessories) et les NDA (New Desk Accessories).

Les CDA sont accessibles à tout moment en tapant au clavier - Control - Esc. Un menu apparaît (c'est là que se trouve le Control Panel qui est un CDA particulier puisqu'il se trouve en Rom): il contient tous les CDA se trouvant en mémoire que vous pouvez utiliser. Il s'agit en général de calepins, de calculatrices, d'utilitaires de macro-commandes (Macromate), ou d'utilitaires divers (The Desktop Manager): mais cela peut être aussi un super-jeu du FTA!

Pour installer un CDA dans ce menu, il existe deux solutions. La première, la moins économique en mémoire et la moins pratique car elle oblige à un reboot, (les accessoires de bureau sont résidents en mémoire) consiste à copier les CDA que vous désirez avoir sous la main dans le sous-catalogue */Desk.Accs* de votre système, puis de rebooter tout le système pour les charger et les installer dans le menu. Vous pouvez par exemple copier le CDA "Beyond.CDA" de la disquette ToolBox-Mag 4 dans votre système afin de pouvoir y jouer à tout moment.

Vous pouvez aussi utiliser un accessoire de bureau permettant d'ajouter en mémoire des CDA et des NDA, puis de les retirer quand vous le jugerez utile. Cet accessoire de bureau existe, il s'appelle "**Fonte/DA Installateur**", c'est un logiciel en shareware de François Uhrich. C'est en fait le seul accessoire qui doive être obligatoirement présent (et activé) dans votre système. Si vous avez le Font/DA Installateur en mémoire, vous pouvez désactiver tous les autres accessoires (sauf le NDA Control Panel) et les mettre où vous voulez, même en-dehors du cata- ▶

logue */System*. Vous les chargerez avec le Fonte/DA Installateur.
"Fonte DA Installateur" est un NDA. On installe les NDA de façon similaire aux CDA, la différence réside dans la manière d'y accéder. En effet, les NDA sont seulement accessibles par le menu des applications en mode desktop. Il suffit de dérouler ce menu et d'y choisir le NDA que vous désirez utiliser. Dans un numéro précédent de ToolBox-Mag, l'accessoire "Traceur" était de type NDA. Si vous risquez d'ouvrir plusieurs NDA en même temps sur votre écran, installez le NDA **Lynx** de P.Desnoues.
Dans le système que vous possédez, il existe un accessoire *CtlPanel.Nda* qui est un tableau de bord graphique: comme c'est un NDA, on y accède par le menu en choisissant Tableau de bord. Cet accessoire travaille avec les fichiers CDev (Control Panel Devices) du sous-catalogue */CDevs*; c'est dans le fichier *Cdev.Data* de ce catalogue que seront rangés les divers paramètres de réglage de votre GS.
Ce nouveau tableau de bord reprend les réglages que vous avez déjà effectués à l'aide du Control Panel. Le NDA se manipule à la souris, il est graphique et utilise de nombreuses icônes... mais n'apporte pas grand chose de nouveau: il est possible d'y configurer la taille de la mémoire tampon utilisée pas GS/OS, ce qui est, c'est vrai, un point important, et c'est par lui qu'il faut désormais passer pour choisir votre imprimante si vous en avez plusieurs.
Chaque élément (représenté par une icône) de ce tableau de bord est associé à un fichier CDev; on peut donc y placer autant d'éléments que l'on veut. Il existe une multitude de CDevs permettant de configurer divers périphériques (carte MIDI, TransWarp) ou d'ajouter un peu de fantaisie au système (extincteur d'écran, chargement musical, beep d'erreur personnalisé...). Ces CDevs se placent dans le sous-catalogue */System/CDevs*. L'installation d'un Cdev ne pose pas de problème; il suffit de le copier dans ce sous-catalogue.

/System/Drivers: votre configuration

Le sous-catalogue */Drivers* contient un certain nombre de gestionnaires de périphériques dépendant de la configuration dont vous disposez. En règle générale, à chaque fois que l'on ajoute un périphérique à l'Apple II GS (lecteur de disquettes, modem, imprimante, carte MIDI), il faut ajouter, c'est à dire copier un driver dans le sous-catalogue */Drivers* de votre volume système. Ainsi, si vous voulez faire reconnaître au système un lecteur 5,25", après la configuration au tableau de bord (voir numéro précédent), il faut recopier sur votre volume système, dans le sous-catalogue */Drivers* le driver *AppleDisk5.25*. C'est pourquoi vous l'avez fait pour constituer votre disque */Mon.Systeme*. La manipulation est la même s'il s'agit d'un modem ou d'un disque dur: seuls les drivers sont différents.
Un driver peut facilement s'activer ou se désactiver (voir plus haut: la marche à suivre est la même que pour les inits). Par exemple, si le Finder vous embête à allumer tout le temps les ampoules de vos disques 5,25, il suffit de désactiver le driver concerné, et de rebooter, pour être tranquille.
Ne désactivez cependant pas le driver appelé *Console.Driver*: ce driver, qui n'est encore utilisé que par GS/OS et par quelques programmes, est un excellent gestionnaire de l'écran-texte du GS, que les programmeurs commencent à découvrir.
Notez aussi que les fichiers appelés *Printer* et *Modem* de ce sous-catalogue ne sont pas des drivers d'imprimante et de modem, mais ceux des ports (1 et 2) du GS. Laissez-les en place.

/System/FSTS: l'indépendance de GS/OS

Le sous-catalogue */Fsts* (File System Translators) contient les modules permettant à GS/OS de travailler avec le système d'organisation des fichiers utilisé par Prodos (*Pro.Fst*), et de reconnaître et de gérer écran et clavier connectés au GS (*Char.Fst*). GS/OS, c'est une de ses forces et de ses supériorités sur énormément d'autres systèmes d'exploitation, est **indépendant de l'organisation concrète des disques qu'il lit**. Il est tout à fait possible d'imaginer d'autres Fsts permettant de lire des disquettes au format HFS Macintosh, au format MS/DOS ou au format... Dos 3.3.
Déjà, GS/OS sait lire les disques compacts au format High-Sierra, et, si vous êtes en réseau AppleShare, vous pourrez naviguer sur les disques Mac avec le Fst *AppleShare*. Il est donc inexact de dire, par exemple, que GS/OS limite les partitions sur un disque dur à 32 Mégas, puisqu'il sait lire des disques HighSierra de 600 Mégas d'un seul tenant: c'est Prodos 8 qui a cette limite, donc le Fst *Prodos* de GS/OS qui gère la compatibilité avec Prodos 8.

/System/Tools: la boîte à outils

Le sous-catalogue */Tools* contient les fameux outils constituant la boîte à outils de l'Apple II GS. Ces outils évoluent à mesure des mo- ▶

difications apportées au système d'exploitation. Ils sont une énorme bibliothèque de routines permettant aux programmeurs de se dégager de tout ce qui est en relation avec l'interface utilisateur. Les outils sont numérotés et chacun d'entre-eux joue un rôle particulier: gestion de la mémoire, gestion des menus déroulants, gestion du son, etc.
Dans ToolBox-Mag, nous avons publié deux outils complets que vous pouvez rajouter à votre système en les copiant dans le sous-catalogue */Tools*. Ces outils permettent aux programmeurs d'utiliser des musiques Music Studio et Soundsmith dans leurs propres programmes. Il en est de même pour le Tool de SynthLab, d'Apple.

/System/Fonts: les polices de caractères

Le sous-catalogue */Fonts* contient les polices de caractères disponibles pour les applications les utilisant. Il y a en Rom une fonte, Shaston.8, qui est utilisée quand aucune autre fonte n'est disponible dans ce sous-catalogue.
Deux fichiers de ce sous-catalogue ne sont pas des polices de caractères. *FontLists* contient la liste des fontes présentes dans le sous-catalogue; ce fichier est modifié par le Font Manager (outil responsable de la gestion des polices de caractères) dès qu'il détecte un changement, ajout ou suppression dans les fontes. *FastFont* est une version prédessinée de Shaston 8; cela permet à QuickDraw II (l'outil responsable de l'affichage) de faire des affichages plus rapides. *FontLists* peut être effacé, il sera de toute façon reconstitué. Si vous supprimez *FastFont*, le GS sera plus lent dans ses affichages, mais continuera à marcher.
Pour installer une fonte, il suffit de la copier dans ce sous-catalogue. Une solution plus élégante consiste à utiliser "Fonte/DA Installateur" (voir plus haut) qui permet aussi d'installer des polices de caractères. Les polices de caractères publiées dans ToolBox-Mag 1 et 3 sont utilisables de cette façon.

Compléments

Le catalogue */Icons* du catalogue principal contient les petites images utilisées par le Finder. Ces icônes sont éditables (par exemple avec Iconed) ce qui permet de personnaliser l'écran (le "bureau") sur lequel on travaille. Chaque programme et chaque fichier peut avoir une image associée. Comme ces images sont insuffisantes pour décrire proprement les fichiers, ce catalogue contient aussi des fichiers *"FType"* contenant, en bon français, des explications sur les divers types de fichiers pour ceux qui savent lire (commande "Lire les Informations" du Finder).
Chaque volume GS/OS peut avoir par ailleurs son propre catalogue */Icons*, contenant des images spéciales pour les fichiers qu'il contient.

Nous n'avons pas parlé ici du programme *Installation* qui se trouve sur la disquette */Utilitaires*. Cet utilitaire fourni avec le système permet, à partir de fichiers de *Scripts*, de réaliser un certain nombre d'installations et de désinstallations: système, drivers, tools, etc.
Chaque configuration ayant un système particulier et personnel, il nous semble préférable d'utiliser un copieur de fichiers performant (Prosel 16) pour réaliser tout le travail sur le système et tous ces modules. La lecture des fichiers du catalogue */Utilitaires/Scripts* est cependant utile pour voir le travail que l'Installeur conseille.
Je ne m'étendrai pas plus sur le disque */Utilitaires*: en-dehors de l'Installeur et d'un utilitaire pour le formatage et la partition des disques durs (*Advanced Disk Utility*), il contient des ressources complémentaires pour le système. Si vous avez une imprimante Epson, une LaserWriter, si vous êtes connecté sur AppleShare, si vous avez une Video Overlay Card, un interface Midi Apple, un lecteur de disques compacts, etc, etc, vous trouverez dans les différents sous-catalogues de ce disque les compléments nécessaires à installer dans votre système (Tools, NDAs, CDAs, CDevs, Drivers, Inits, etc: toutes ces petites bêtes vous sont désormais familières).
Il faudrait d'ailleurs ajouter à cette liste les composants du système qui vous sont fournis avec les périphériques non-Apple (drivers de disques durs, de carte SCSI, de carte Midi, accessoire de Quickie, etc), ainsi que vos inits, accessoires, fontes, tools. Tout cela fait aussi partie du système: il n'existe pas **le** système, mais seulement **votre** système.

Le vrai système: le vôtre

Au fond, cela signifie en fait que **le véritable système du GS (c'est-à-dire le vôtre) fait bien plus de 800k**. Si vous incluez toutes les fontes, les Inits, les accessoires, les drivers et Cdevs, activés ou désactivés, vous arriverez très facilement à un système de deux ou trois mégas, et c'est tout-à-fait normal.
Le disque */Mon.Systeme* n'était qu'un mini-système qui a pu nous servir de démonstration, mais qui n'est pas sérieusement utilisable dans ▶

le monde réel. Il est incapable de contenir ne serait-ce que l'ensemble des accessoires, fontes, outils, inits, etc, que ToolBox-Mag vous a d'ores et déjà fournis!

Un disque dur ou tout autre lecteur de 20 Mégas au moins est désormais une nécessité incontournable sur l'Apple II GS, comme d'ailleurs sur tous les ordinateurs personnels (j'ai dit "ordinateur", pas "console" ni "casserole"). Les disquettes, qu'il s'agisse des disquettes système ou des disquettes ToolBox-Mag, ne sont plus qu'un moyen de communication entre utilisateurs.

Vous devinez donc quel sera le sujet d'un des prochains "GS pour débutants": il ne sera déjà plus pour "débutants" (vous ne l'êtes désormais plus!), et il concernera la gestion des mémoires de masse.

GS/OS, un système hors pair

Vous voilà donc à même de faire maintenant, en connaissance de cause, votre propre système, spécial pour votre configuration. Vous pourrez ensuite utiliser tranquillement les programmes de ToolBox-Mag. A vous de jouer.

Oui, il y a un peu de travail qui vous revient: c'est le prix de la puissance et de la souplesse du système du GS. Avec un peu de pratique, vous passerez très vite du désarroi du débutant (*"c'est quoi, une Init?"*) au sourire indulgent du connaisseur (*"tiens, le Finder du Mac ne permet pas de désactiver une Init ou un accessoire?"*).

Quand on fait un travail professionnel en informatique, on est bien forcé de fréquenter plusieurs systèmes d'exploitation, au moins ceux des machines les plus diffusées par ailleurs, à savoir MS-Dos et HFS-Macintosh. Et on ne peut que se rendre à l'évidence: si l'on met à part la question du multitâches, que GS/OS ne pratique pas, **GS/OS est une référence pour les autres systèmes d'exploitation** d'ordinateurs personnels (le Mac ayant le mérite, avec son futur système 7, de prendre explicitement modèle sur lui).

En apprenant à maîtriser GS/OS, vous ne vous pliez pas à des tics aberrants, des limites historiques, des habitudes arbitraires ou des bidouilles incompréhensibles.

Non seulement vous apprenez à piloter votre machine, mais vous vous exercez à utiliser **un des meilleurs logiciels existant en ce moment**. Quand on s'est formé sur GS/OS, on devient un utilisateur exigeant: cette exigence des utilisateurs est un facteur essentiel du progrès de l'informatique personnelle. ■

Une manière sûre d'utiliser l'Installeur.

Si Eric vous déconseille d'utiliser l'Installeur, c'est que celui-ci vous fait travailler "en aveugle" (ce n'est pas une critique: il est précisément fait pour cela). L'Installeur écrase sans prévenir les fichiers existants s'ils ne lui plaisent pas (si votre */System/Start* était Select, Prosel ou Wings, vous avez toutes les chances de retrouver le Finder à sa place!), il vous laisse l'essentiel du 5.04 américain si vous l'aviez mis auparavant, etc. Dans les systèmes ultra-personnalisés (super-bidouillés) que tendent à devenir les catalogues */System* de nos disques durs, le résultat peut-être imprévisible, indémarrable et/ou indémêlable.

Pourtant l'Installeur est bien pratique, parce qu'il travaille tout seul et parce qu'il n'oublie jamais, lui, les petits fichiers cachés dans des recoins de la disquette */Utilitaires*. Si vous vous décidez pour la Laser, par exemple, c'est la meilleure formule pour installer l'ensemble des fichiers nécessaires.

Voici donc un truc tout simple que j'utilise, et qui marche pour tous ceux qui ont un disque dur. Il évite même de fabriquer la disquette */Mon.Systeme* d'Eric:

• Bootez la disquette */Disque.Systeme* du 5.04 français. Le Finder devrait reconnaître votre disque dur.

• Renommez (c'est ça le truc) le catalogue */System* de votre disque dur en */System.1*.

• Introduisez alors la disquette */Utilitaires*, et lancez l'Installeur. Dîtes-lui d'installer le maximum sur votre disque dur. Comme il ne trouvera pas de catalogue */System*, il va en créer un nouveau sans rien détruire de l'ancien (qui marche, c'est sûr). Il copiera tout ce qu'il faut, et vous fera un système qui devrait, en principe, marcher.

• Redémarrez alors sur votre disque dur: si ça marche, il ne vous restera plus qu'à recopier (je dis bien copier, pas déplacer) de */System.1* vers */System* tout ce que vous aviez rajouté dans votre ancien système.

Au moindre pépin, il suffit de renommer */System* en */System.2* et */System.1* en */System* pour retrouver un dur qui marche: vous n'êtes jamais en panne. Vous pourrez toujours détruire */System.2* et recommencer l'opération. Quand vous aurez un nouveau système complet, vérifié, et qui ne plante pas, il suffira de détruire */System.1* avec tout son contenu.

JYB

MuSeq

vers un séquenceur avec l'outil SynthLab

Jean-Pierre Charpentier

*[NDLR: Sérieux problème avec cette seconde partie de l'étude de Jean-Pierre Charpentier sur le Tool35: comme toujours, il vous faut le Tool35; cette fois-ci, il vous faut aussi un interface Midi et un clavier: c'est normal, c'est le but du jeu. Cela devrait s'arrêter là: hélas, il vous faut aussi… le système 5.04 version américaine. En effet, il y a un pépin, **un problème quelque part dans le 5.04 français** qui empêche **MuSeq** de tourner, alors qu'il marche parfaitement sous système US. Il nous a semblé que le coupable était le petit fichier **Tool.Setup** du sous-catalogue **/System/System.Setup**. Nous avons donc mis sur la disquette ToolBox-Mag le fichier **Tool.Setup** du système 5.04 américain. En remplaçant, dans votre système, le Tool.Setup français par le Tool.Setup US, ça devrait marcher (ça marche chez nous). Mais nous ne garantissons rien…*
*Ca n'est pas si grave: **MuSeq** est surtout conçu pour être utile aux programmeurs, en démontrant l'utilisation de la partie Midi du Tool35. **Ce sont les sources qui sont le principal**. Du coup, puiqu'il ne marche à coup sûr qu'avec le système américain, nous avons aussi laissé les ressources en anglais. A vous de franciser avec Genesys. Quand même, toutes nos excuses. Et **si quelqu'un peut trouver le bug et son remède, il rendra service à tous…**]*

A propos de SynthLab

Tout d'abord quelques précisions pour être sûr que tout le monde parle bien le même langage. Le package appelé SYNTHLAB, tel qu'il est vendu par l'APDA, regroupe:
• *MIDISynth*, qui est le Tool35
• *synthLab*, (sans majuscules) qui est un programme démontrant l'utilisation du Tool35
• *MIDI.CDev*, qui est un fichier utilisé par le Tableau de bord. A chaque fermeture du CDev MIDI.CDev, autrement dit à chaque fois que vous modifiez votre configuration Midi, un fichier de configuration nommé *Midi.SetUp* sera automatiquement sauvé dans le sous-catalogue */System/Drivers*. Ce fichier est utilisé pour indiquer au Tool35 quel driver Midi doit être chargé, et à quel endroit.

Le Tool35 pouvant être trouvé avec certains logiciels tout nu, sans le CDev Midi, voici le format du fichier généré */System/Drivers/Midi.SetUp* (type BIN, type auxiliaire $0000, longueur 37 octets).

• Word: Slot externe/Interne (0/1)
• Word: Numéro de slot (1-7)
• Str[32]: Chaîne Pascal du nom du fichier Driver MIDI (nom de fichier, pas chemin d'accès complet: le préfixe supposé est */System/Drivers/*).

Présentation de *MuSeq*

Le programme de ce numéro s'appelle **MidiSynth.MuSeq** car c'est la base, d'où le Mu, d'un séquenceur, Seq. Il ne faut surtout pas trop lui en demander, synthLab est là pour ça.
Pour comprendre toute la puissance du Tool35, il faut déjà posséder quelques connaissances Midi. Or un article couvrant le standard Midi en entier remplirait tout un numéro de ToolBox-Mag. Heureusement on trouve maintenant facilement des livres *en français* sur Midi dans n'importe quelle (bonne) boutique d'instruments de musique.
Mais commençons par le commencement: au démarrage de *MuSeq*, la séquence *Bee.Seq* est chargée en mémoire. Le fichier *Bee.Seq* **doit** donc obligatoirement se trouver *dans le même catalogue que MuSeq*, ainsi d'ailleurs que *Comb.Bnk* et *Comb.Wav*, respectivement fichier d'instruments et fichier d'onde référencés dans *Bee.Seq*.
Dans le programme précédent, on pouvait choisir sa séquence facilement à l'aide d'un *_SFGetFile*. Maintenant le nom du fichier séquence est inscrit en ressource. Ca ne sert à rien, sauf… à montrer que c'est faisable. Vous pouvez éditer ce nom à l'aide d'un éditeur de ressources comme Genesys. Attention à ce que le nom ne dépasse pas 15 caractères et à ce que le fichier (et ses fichiers associés) existent réellement ▶

dans le catalogue de *MuSeq*.
La ressource contenant ce nom est de type *rC1InputString* ($8005) et son ID est *Synth_Seq_C1InputString_I* $00000001). Pour connaître les fichiers associés à une séquence il faut utiliser synthLab (ou BlockWarden!).

MuSeq se compose principalement de quatre fenêtres:
WD01 = fenêtre de commande du magnétophone (Stop, Start, Avance rapide, Retour forcé au début).
WD02 = fenêtre du choix du canal de base et du mode.
WD03 = fenêtre de validation des pistes en lecture et en enregistrement.
WD04 = fenêtre d'assignation des pistes par rapport aux canaux Midi (donc aux instruments).

Certains paramètres sont constants:

Le tempo est récupéré dans le fichier séquence.
*_SetTempo (SEQU_PTR.Tempo*2-10)*

Le battement est aussi récupéré dans le fichier séquence.
_SetBeat (SEQU_PTR.Ticks_Per_Beat)

Toutes les pistes sont validées en lecture (par défaut).
_SetPlayTrack (Track Number (0 -> 15), True)

Les événements de chaque piste ne subissent pas de désassignation de canal, donc ils joueront par défaut leur instrument.
_TrackToChan (Track Number (0 -> 15), EOS=$FFFF)

J'ai choisi d'envoyer la sortie du séquenceur en permanence vers le synthétiseur **et** l'interface Midi.
_SetTrackOut (Track Number (0 -> 15), PathValue) avec *PathValue=0.*

L'assignation des instruments est directe.
_SetInstrument ((BANK_PTR.Inst_Record_Offset(0 -> 15)), Inst Number(0 -> 15)).*

Notez que les pistes et canaux Midi sont numérotés de 1 à 16 pour l'utilisateur et de 0 à 15 ($F) pour le programmeur (exception faite du flag d'enregistrement *RECDFLAG* où on met le numéro de piste **+ 1**, car pour la piste 1 le flag serait à zéro).

Première fenêtre

Dans la première fenêtre (*WD01*), il n'y a pas de contrôle sur la limite du bouton *Forward*, donc à utiliser avec précautions.
Le bouton de *RAZ* ramène à zéro dans tous les cas, que l'on soit arrêté, en lecture ou en enregistrement.
Comment peut-on enregistrer alors qu'il n'y a pas de bouton *Record*? A chaque appui sur le bouton *Play* on regarde si *RECDFLAG* est différent de zéro, c'est à dire si une piste est marquée en enregistrement. Si oui, on envoie un *Play* avec *Record* et c'est tout!
Le démarrage en enregistrement validera également automatiquement le métronome, qui battra le tempo.

Deuxième fenêtre

La fenêtre numéro 2 (*WD02*) permet de choisir le mode d'enregistrement du séquenceur et le canal de base. Pour ce dernier, on peut bien sûr choisir entre 1 et 16. Le choix du mode agit comme un filtre pour les données qui arriveront par exemple d'un clavier Midi.
En mode *Omni*, tous les événements Midi sont affectés au canal de base.
En mode *Poly*, les événements sont triés. Ceux correspondant au canal de base sont enregistrés sans modification. Les autres vont directement à la poubelle, et sont perdus corps et âme!
En mode *Multi*, le filtre est déconnecté et tous les événements sont enregistrés sans aucune modification. Ce mode est rarement utilisé, car en général on préfère avoir tous les événements d'une piste pointant sur le même canal, donc le même instrument.

Troisième fenêtre

La fenêtre 3 (*WD03*) permet de fixer chaque piste en lecture et en enregistrement. Pour la lecture, toutes les combinaisons sont possibles entre les seize pistes, donc l'utilisation de cases à cocher est logique.
Une seule piste pouvant être en enregistrement, *a priori* le choix de boutons radio s'imposait. Mais on peut n'avoir aucune piste en enregistrement, ce qui représente un cas non prévu dans la gestion des boutons radio. J'ai donc utilisé des cases à cocher, avec une gestion un peu particulière: à chaque clic sur une case, si celle-ci passe de 0 à 1, j'efface toutes les autres cases; si elle passe de 1 à 0, je ne touche pas aux autres.
Tout ce qui apparaît dans *WD03* est fait de contrôles, il n'y a aucun dessin direct dans la fenêtre, sauf pour le fond.
Notez que l'utilisation de *StaticText* pour chaque numéro est une erreur. A partir du moment où les différentes parties à afficher sont alignées, on a fortement intérêt à utiliser un seul *StaticText* de longueur convenable. C'est moins lourd et plus facile à gérer (voir *WD04*). ▶

De plus Genesys est encore buggé en version 1.2 (erreur $0204 = Invalid Operation on Locked Block). J'ai pu vérifier que ce bug était corrigé en version 1.2.4.

Quatrième fenêtre

Pour la fenêtre 4 (*WD04*), j'ai choisi d'utiliser des boutons radio au lieu de scroll-bars, **ce qui est une grave erreur** du point de vue de la lourdeur et du temps de réponse du programme!
Même avec ma TransWarp 8 Mhz turbo-injection, ça rame tellement que j'ai été obligé de prévoir un message d'attente pendant la construction des fenêtres, de peur que l'utilisateur ne reboote, pensant être planté!
Vous pouvez donc, avec cette fenêtre, rediriger les événements de chaque piste vers un canal Midi particulier, ou laisser le réglage sur le canal zéro qui signifie "pas de redirection". Dans ce cas les événements jouent sur leur canal propre, c'est à dire celui sur lequel ils ont été enregistrés.
On peut considérer cette fenêtre comme une *assignation des pistes en sortie* alors que l'ensemble canal de base-mode permet une *assignation en entrée.*
Si vous branchez un synthétiseur/expandeur multi-timbral, vous aurez aussi la possibilité de régler l'assignation en sortie sur votre appareil. Par exemple, avec la séquence *Bee.Seq* (Vol du Bourdon), le canal numéro 1 est celui du *DrumKit*. Si vous envoyez toutes les pistes sur le canal 1, toutes les notes seront jouées avec cet instrument, et vous obtiendrez un beau solo de batterie.

Notes de programmation

Du point de vue de la programmation, j'ai mis tout ce que j'ai pu en ressources. Cela vous permettra de faire une version française de ***MuSeq*** si vous le souhaitez (Toolbox-Mag 4 et son *DefaultFile* français sont arrivés trop tard, Genesys avait déjà démarré une application en anglais).
Avec l'arrivée des *super-contrôles*, est aussi arrivée une nouvelle forme de gestion des contrôles. Les contrôles étant référencés par un handle de 32 bits difficile à manier en assembleur, je faisais comme tout le monde et affectais un numéro dans le champs *RefCon* de chaque contrôle. Maintenant c'est inutile, car chaque contrôle posséde son propre ID.
Mais comme comme les routines d'accès se font toujours par handle, il est intéressant de se fabriquer de petites routines que l'on va pouvoir copier/coller tous azimuts.

Par exemple, pour rendre inactif le Contrôle 2, il faut d'abord récupérer le handle:

```
pushlong    #0          [on fait de la place]
pushlong    WD01        [ctlWindowPtr]
pushlong    #Control_2  ctlID
_GetCtlHandleFromID
pulllong    LONGBUFF
pushword    #255        [inactif]
pushlong    LONGBUFF
_HiliteControl
```

On peut comprimer les deux opérations en une seule, en jouant sur la pile:

```
pushword    #255        [inactif]
pushlong    #0          [on fait de la place]
pushlong    WD01        [ctlWindowPtr]
pushlong    #Control_2  ctlID
_GetCtlHandleFromID
_HiliteControl
```

C'est plus compact et plus pratique pour le copier/coller. De plus, si *ctlWindowPtr* correspond à la fenêtre de premier plan, on peut mettre *Nil* comme valeur.

A noter: le *refCon* de chaque fenêtre doit être fixé au moment de l'appel *_NewWindow2*, car celui de la ressource template n'est pas pris en compte. Voyez la Note Technique IIGS 24, révision Janvier 91.

Comme le Tool35 tourne en tâche de fond, l'appel *_SetCallBack* permet de récupérer certains événements. Dans ***MuSeq***, je récupère seulement l'arrivée d'un packet Midi complet (*PACK_IT*), pour faire flasher la bordure, et la fin de la séquence (*END_SEQ*), pour arrêter la lecture/enregistrement.

Pour les nombreux boutons radio (17*16 !) de *WD04* j'ai utilisé Rez et DeRez. Cela m'a permis d'assigner la même chaîne Pascal vide pour les titres de tous les boutons (ID=$7777777).

L'appel *_SetVelComp* sert à ajouter du volume aux notes envoyées par le clavier Midi.

L'appel *_GetMSData* donne l'adresse de la page zéro utilisée par le Tool35. On pourra y pêcher des informations intéressantes à n'importe quel moment.

Conclusion

Terminons par une appréciation: si le Tool35 est plus fiable pour sa partie Midi que le MidiManager, ce n'est pas encore la panacée. On se retrouve de trop nombreuses fois planté dans le driver Midi! Même avec tous les efforts pour décharger le driver proprement, on rend souvent visite au monitor (ou au debugger). ■

Introduction à la programmation de l'Apple II GS :

La gestion des événements

Jean Destelle

Dans cette série d'articles, la programmation du GS est expliquée à l'aide d'exemples en Pascal. Toutefois, comme le vocabulaire de la Toolbox est universel, et utilisé de la même façon par les autres langages comme l'Assembleur et le C, il intéressera tout lecteur désireux de mieux comprendre la programmation du GS.

La boucle des événements

Un programme du type "DeskTop", fonctionne grâce à l'emploi d'une *boucle d'événements*. Cette boucle, décrite en général dans une procédure portant le nom de *MainEventLoop* tourne en permanence tant que le programme fonctionne.
Après avoir mis en place tous les éléments du "bureau" et principalement les menus déroulants qui permettront d'appeler les diverses fonctions de l'application, on lance la boucle d'événements. Sans cesse, infatigable, elle va répéter un circuit très complexe, vérifiant tout au passage.

Le vrai patron de ce circuit touristique permanent, c'est **TaskMaster**, un serviteur incomparable (celui-là, les programmeurs sur Mac nous l'envient).

TaskMaster

TaskMaster fait un boulot formidable. Entre autres, il fait tout seul presque tout ce qui a été décrit dans les ouvrages spécialisés sur la "boite à outil du GS" à ses débuts, et qui a rebuté nombre d'entre nous.
A chaque tour, il demande à EventManager s'il s'est passé quelque chose, en utilisant la fonction *GetNextEvent*. EventMgr, lui n'a pas cessé de guetter des événements. Tous ceux qui se sont produits depuis le dernier passage de la boucle sont notés à la queue-leu-leu dans une file d'attente. S'il y a un "événement" dans cette file d'attente, EventMgr l'indique à TaskMaster, en lui donnant toutes les informations possibles. Cet événement peut être l'enfoncement d'une touche, un clic de la souris, ou toute autre chose.
Suivant les instructions que vous lui aurez données, TaskMaster fera tout le nécessaire à votre place, ou seulement très peu, si vous vous réservez le plaisir par exemple de faire vous-même la gestion complexe d'une barre de défilement, ou d'un autre système vicieux du même acabit. Il s'occupera même de redessiner les fenêtres en cas de besoin.

Avec l'outil TextEdit, TaskMaster vous donne un tel coup de main que cela semble miraculeux. Avec le système 5.0, Apple a vraiment simplifié la vie des programmeurs GS. Vous en avez un exemple dans le petit programme *MiniDeskTop* dans lequel vous pouvez admirer comment TaskMaster, tout seul, sans que nous ayez eu besoin d'écrire la moindre instruction à ce sujet, gère le texte à l'intérieur d'un contrôle TextEdit. Regardez-bien le source. Il n'y a rien à chercher! **TaskMaster fait cela tout seul.**

NB: *MiniDeskTop* est un programme de démonstration, très court, utilisant l'unit *AuxSelect1* de *Sélect*, destiné à servir à la fois de début standard et de base à nos explications sur l'emploi du Pascal. Vous le trouverez dans le catalogue */MiniDeskTop* du disque.

TaskMaster gère ainsi tous les contrôles, aussi bien les *boîtes à cocher* que les nouveaux menus *Pop-Up*. Et les fenêtres? Aussi! Cela n'a l'air de rien, mais tous ces mouvements de fenêtres, actionnements de barres de défilements, zooms, etc, il aurait fallu, sans TaskMaster, les obtenir en appelant toutes sortes de fonctions successives, un vrai casse-tête.

Notez que puisque TaskMaster gère les événements, et qu'il existe un outil "EventManager" dans la boîte à outils, vous trouverez tout naturellement TaskMaster... dans l'outil "WindowManager"!

Le Task-Mask

L'ordre de mission que vous devez donner à TaskMaster est un "masque", le "Task-Mask". C'est en fait un "bit-flag" énorme, qui comporte 32 bits, avec des tas d'instructions pour faire et ne pas faire certaines tâches. Ce masque est l'un des champs de l'*event-record*, une structure de données dans laquelle EventManager note tout ce qui arrive.

Octets, bits, et bit-flag

Vous le savez, chaque octet se compose de 8 éléments binaires (bits) successifs, qui ne peuvent prendre que la valeur 0 ou la valeur 1. Par exemple l'écriture en binaire du nombre $1F est 00011111, que vous pouvez décomposer en deux quartets (*nibbles*): 0001, équivalent au chiffre hexadécimal 1, et 1111, équivalent au chiffre hexadécimal F.
Un *Bit-Flag* est un nombre dont chacun des bits séparément sert à enregistrer une donnée. Sur un octet on peut ainsi noter jusqu'à 8 conditions indépendantes. Le mot *flag* (drapeau) est à prendre ici au sens de "panneau indicateur".
Les bit-flags sont très utilisés par Apple dans la Toolbox. Ils permettent de résumer en un seul octet jusqu'à 8 conditions différentes. Dans le GS, ordinateur 16 bits, vous rencontrerez surtout des bit-flags ▶

de deux octets.
Attention, les bits se numérotent de droite à gauche. Dans l'exemple du nombre $1F, la numérotation est la suivante:

Numéro d'ordre du bit:	7 6 5 4 3 2 1 0
Valeur de chaque bit :	0 0 0 1 1 1 1 1

Autre exemple, le bit-flag *Attributes* utilisé par la fonction *NewHandle* de MemoryManager est un entier de deux octets. Supposons que la valeur de cet entier soit $8018:

	octet de poids fort	octet poids faible
Numéro du bit:	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
Valeur du bit:	1 0 0 0 0 0 0 0	0 0 0 1 1 0 0 0

la mise à 1 du bit 15 signifie: le bloc de mémoire est verrouillé (*Attrlocked*); la mise à 1 du bit 4 signifie: le bloc de mémoire ne doit pas chevaucher la limite d'un banc de mémoire (*Attrnocross*); la mise à 1 du bit 3 signifie: on ne doit pas utiliser la mémoire spéciale (*AttrNoSpec*).

Utilisation de constantes pour donner des valeurs aux bit-flags. Continuons avec l'exemple ci-dessus. Apple définit, dans le Toolbox Reference, parmi d'autres, les constantes:

attrLocked = $8000;
attrNoCross = $0010
attrNoSpec = $0008;

Dans l'écriture d'un source, on peut utiliser ces constantes pour définir un bit-flag, en les additionnant. En effet, *attrLocked* + *attrNoCross* + *attrNoSpec* = $8000 + $0010 + $0008 = $8018. On peut donc écrire directement $8018. Mais il est beaucoup plus parlant d'écrire *attrLocked* + *attrNoCross* + *attrNoSpec*, ce que les compilateurs comprennent très bien.

Bits-flags et masques. Un "masque" est un nombre généralement écrit en hexadécimal sur 1,2 ou 4 octets. Ce nombre servira, au moyen d'une opération très rapide pour le processeur (Opération "ou" , "et" ou bien "ou exclusif" avec un autre nombre), à fournir un résultat déterminant pour la suite des opérations. Le "masque" aura ainsi "décodé" le nombre.
Par exemple, si on doit lancer une procédure quand le nombre entier x est pair (c'est-à-dire que son bit numéro 0 est à 0), il suffira de faire une opération "et" entre la variable x et le masque $0001. Si le résultat de cette opération est 0, c'est que x est pair. Si le résultat est 1, c'est que x est impair. On se sert beaucoup de "masques" pour "filtrer" des codes, par exemples pour éliminer des caractères indésirables lors de l'impression.

Ecrivons un programme

Reportons-nous au source de *MiniDeskTop*. Au début de la *MainEventLoop*, vous lisez le traditionnel:
gMainEvent.wmTaskMask:= $001FFFFF;
gmainEvent est le nom de l'*eventRecord*, qui doit être une variable globale; *wmTaskMask* est ce fameux masque. La valeur *$001FFFFF* est celle qu'on obtient en répondant oui à toutes les questions sur ce qu'il doit faire. Il signifie: faites tout à ma place.
Muni de ce blanc-seing, TaskMaster gérera, quand vous ferez appel à lui, les menus, les fenêtres, les contrôles, et tout et tout, sans que vous ayez à intervenir, du moins tant que ne vous est pas signalé un événement que vous voulez traiter vous-même.

La boucle principale

Dans la procédure *MainEventLoop*, il y a ensuite l'assignation de la valeur booléenne *false* à la variable *done* (*done* veut dire "c'est fini"). Puis la boucle débute par *repeat*, et se terminera à *until* si la condition *done* est vraie, ce qui pourra être le fait d'une décision extérieure mettant fin à l'application, en général une procédure *DoQuit*.
Aussitôt après *repeat*, vous trouvez l'instruction:
code:= taskMaster($FFFF, gmainEvent);
Cette instruction est l'appel à TaskMaster, et lui donne l'ordre de regarder dans l'*event-record* les circonstances de l'événement qui s'est produit, et d'agir en conséquence.
TaskMaster fait donc son travail, qui est parfois très compliqué. Puis il vous rend compte, en mettant à votre disposition, à l'intérieur de l'*event-record*, diverses informations où vous pourrez allez les chercher quand vous en aurez besoin: le genre d'événement, et aussi des données importantes qui varient suivant le genre de l'événement.
Il vous donne en premier lieu une indication générale sur ce qui s'est produit, un nombre qui se trouve dans la variable *code*. Suivant la valeur indiquée, vous saurez qu'il s'agit, par exemple, de l'actionnement d'un menu, ou d'une touche de clavier, ou d'un clic dans une fenêtre, ou dans un contrôle. Si la chose est de nature à vous intéresser, vous vous en occuperez en appelant des procédures que vous devrez écrire; si elle ne vous intéresse pas, vous ne vous en occupez pas, TaskMaster le fera à votre place.
Les valeurs possibles du code d'événement sont des constantes dont les noms décrivent l'événement. Il y a un grand nombre de possibilités.
Par exemple *wGoAway* indique qu'on a cliqué dans la case de fermeture d'une fenêtre. Vous devez alors appeler une procédure *doclose* qui fermera aussi bien un NDA qu'une des fenêtres de l'application. De même *WinMenuBar* indique qu'on a actionné un menu et cliqué dans l'un des choix proposés par le menu. Vous appelez alors une procédure pour agir en conséquence: dans *MiniDeskTop*, elle s'appelle *GererMenu*. Ou encore *WinControl* indique qu'un événement s'est produit dans l'un des contrôles de la fenêtre sélectionnée: on appelle alors *GererControles*.

La boucle d'événements de *Sélect*

Dans ***Sélect*** on s'occupe particulièrement du cas *WinContent*, qui indique un clic dans le contenu d'une fenêtre. Cela parce que les cases de lancement des programmes ne sont pas des contrôles, mais de simples rectangles dessinés dans la fenêtre. Il faut donc prendre nous-même ce problème en charge en appelant la procédure *TraiterWincontent*.
Notez aussi que nous prenons en compte le code *inNull* dont on ne se sert pas d'habitude, et qui indique tout simplement qu'aucun événement ne s'est produit. Nous nous en servons pour appeler une procédure *Economiseur*. Cette procédure regarde depuis combien de temps il ne s'est rien passé, et si plus de deux minutes se sont écoulées et que ***Sélect*** n'est pas en mode "édition", elle provoquera la mise au noir de l'écran du GS. Si le code fourni par TaskMaster n' a pas été retenu, tant pis; on repart pour un nouveau tour de boucle.

▶

Les nouvelles fenêtres de dialogue

Dans le courant du déroulement de l'application, lorsque le menu appelle certaines fonctions, il devient nécessaire de questionner l'utilisateur, au moyen d'un "dialogue", afin de prendre les consignes pour la suite.
Les dialogues sont (ou plutôt étaient) régis dans le GS par DialogMgr, un outil extrêmement pratique. Malheureusement les fonctions de cet outil n'ont pas encore, à cette date, été mises à jour pour l'emploi de ressources, de sorte que le dessin des fenêtres et des contrôles doit toujours se faire avec l'ancienne (fastidieuse) méthode.
On n'emploie plus maintenant pratiquement que des fenêtres ordinaires, créées par la nouvelle fonction *NewWindow2*, et comportant des contrôles inclus soit au moyen de *NewControl2*, soit au moyen d'une liste de contrôles incluse dans la ressource de la fenêtre (ce qui est notre cas). Mais subsiste toujours la notion très importante de dialogues modaux et non modaux.

Dialogues modaux et non-modaux

Un dialogue est dit *modal* quand il force l'utilisateur à répondre à ses questions, sans lui laisser la possibilité de déclencher une autre action par un menu, une autre fenêtre, ou autre chose. Au contraire, il est dit *non-modal* (*modeless*) c'est-à-dire normal, si on peut ne pas y répondre tout de suite, et agir auparavant sur le déclenchement d'une autre fonction. Ce dernier cas constitue la situation habituelle, hautement recommandée par Apple, qui rend toutes actions directement accessibles à tout moment.
Par exemple, une fenêtre d'alerte utilisant la fonction *AlertWindow*, dont vous rencontrerez plusieurs exemples dans les sources, est un dialogue *modal*. On est obligé de répondre avant de continuer. En revanche, la fenêtre-écran de lancement de *Sélect* constitue un dialogue *non-modal*, car à tout instant, on peut utiliser les autres fonctions du menu, ou bien employer un accessoire de bureau.
Dans le petit programme *MiniDeskTop*, il n'y a pas (encore) de dialogue modal. Vous pouvez donc, tout en expérimentant les deux fenêtres, appeler les autres fonctions du menu. Il n'en est pas de même dans l'application *Sélect*, qui comporte de plus nombreuses fonctions.
Dans *Sélect*, nous utilisons plusieurs dialogues modaux. Pour les rendre indépendants des autres fonctions de l'application, nous utilisons une autre boucle, mise en route après que la fenêtre du dialogue, avec ses contrôles, ait été dessinée. Cette *boucle secondaire* va tourner jusqu'à ce que, le dialogue étant achevé, et les opérations correspondantes effectuées, on puisse revenir à la boucle principale.
La procédure *ModalLoop* matérialise cette boucle. Elle est construite à partir des mêmes éléments de base que la boucle principale: *gmainEvent*, et TaskMaster. Mais cette fois, nous ne donnons pas toutes les permissions à TaskMaster. Nous restreignons son rôle à quelques actions bien précises, qui lui permettront seulement de gérer les contrôles dont nous avons besoin.
La valeur retenue pour le masque est $00132006. Ne sont pas mis à 1, entre autres, les bits concernant le menu, ou la sélection éventuelle d'une autre fenêtre. De cette façon, le dialogue devient modal. Nous n'utiliserons que ce qui concerne les contrôles: action des touches du clavier transmises aux contrôles *LinEdit* qui permettent les entrées de textes et de nombres, et clics de la souris dans les boutons. La variable *done* étant réservée à la boucle principale, nous employons une autre variable *Termine* pour mettre fin au bouclage. Seuls les boutons de sortie seront capables de provoquer la fin de la boucle modale, par l'intermédiaire de la variable *Termine*.
En fin de boucle, avant de sortir, il ne faudra pas oublier de redonner à *TaskMask* sa valeur initiale, sans quoi plus grand chose ne marcherait. Par la suite, TaskMaster pourra reprendre sa tournée avec les pleins pouvoirs.

Nous terminons ici cet exposé sur la gestion des événements par le GS. Si vous savez gérer la mémoire et les événements, vous pouvez commencer à vous lancer, et écrire des programmes GS qui tournent... ■

C'est bien un Apple II

Certains programmeurs n'aiment pas ce que fait Joël Desnoues dans ToolBox-Mag 4 pour éviter des crampes à Hubert. Ils n'ont pas envie que les utilisateurs puissent interrompre leur programme et passer dans le tableau de bord, histoire de tricher avec Visit Monitor II.
Comme Apple a eu la gentillesse de publier le vecteur du Control Panel, ils mettent donc un CLC/RTL à la place de ce vecteur, pour empêcher l'accès au tableau de bord par ⌘-Control-Escape.
Mais Apple tient à cette possibilité, puisqu'il a mis le tableau de bord en Rom. Et quelqu'un, chez Apple, déteste être privé de son tableau de bord et de son Visit Monitor: dans les Roms 03 du GS, le coup est prévu.
Il suffit de faire Shift-⌘-Control-Escape au lieu de ⌘-Control-Escape, et on se retrouve dans le tableau de bord (on ne passe plus par le vecteur). C'est bien un Apple II...
Bien entendu, il y a des techniques plus sohistiquées pour désactiver le Control-Panel. Mais il faut être vraiment méchant.

– Bon alors, dans Photonix II, ça marche, ce truc, ou pas?
– Essayez...

Olivier Goguel

Introduction à la programmation de l'Apple II GS :

Introduction aux ressources, première partie

Jean Destelle

Vivent les ressources !

Vous savez que les fichiers du nouveau type "étendu" introduits par GS/OS ressemblent à ceux du Mac, et comportent deux parties différentes : un segment de données tout à fait similaire aux fichiers des anciens types (appelé "data-fork"), et un segment de ressources, (appelé "res-fork") destiné à contenir les ressources.
Prenez par exemple le programme *MuSeq* de J.P. Charpentier dans ce numéro: il consiste en un fichier étendu. Le code, c'est-à-dire le programme lui-même, est enregistré dans le segment données. Le segment ressources reçoit tout ce qui est susceptible d'être souvent modifié, comme les titres (afin de pouvoir les traduire), les textes des alertes, ainsi que tous les blocs de paramètres qui définissent les caractéristiques : emplacements, formes, formes et couleurs des menus et des fenêtres, et des éléments de contrôle et de dialogue.
Les avantages de ce procédé sont évidents. On peut traduire facilement un programme dans une autre langue. On peut en modifier la présentation sans intervenir dans la partie "code" toujours délicate à modifier. On peut constituer une librairie d'éléments standard transférables simplement d'un programme à l'autre sans modification, ou avec des modifications très faibles. Les ressources n'encombrent pas la mémoire, car elles n'y sont transférées qu'au moment où on en a besoin.
Avec les ressources, la plus grosse partie de ce que nous écrivions jusqu'à maintenant sous forme de source dans les programmes peut être constituée à partir de données standard, ou déjà précédemment utilisées. Cela très rapidement, au moyen de logiciels spéciaux: les éditeurs de ressources. Il devient possible, dès maintenant de diviser par 2, 3 ou même 4 le travail de programmation.

Les ressources: des données structurées

Les "ressources" sont en fait tout simplement des *données structurées,* et "Resource Manager" un outil très performant pour la manipulation de ces données.
Qu'est-ce qu'une donnée structurée? Tout simplement un ensemble de données élémentaires variées regroupées sous un même nom. Les données élémentaires peuvent être n'importe quoi, à condition de pouvoir les traduire en une succession d'octets: des chiffres, des codes d'instructions de programme, des textes, des dessins, des échantillons sonores, les caractères d'une fonte, etc.
Ainsi votre numéro de Sécurité Sociale est une donnée structurée, de même que votre code postal: on peut les manipuler c'est à dire les lire, les écrire, les déplacer, sans avoir à connaître le détail de leur structure interne. Mais aussi on peut en extraire les données internes séparément.
L'état d'esprit du GS, c'est de considérer cet ensemble de données comme une structure quelconque, un "Record" au sens du langage Pascal. De ce fait, nous pourrons utiliser des outils très généraux pour manipuler ce "record" comme les autres. Pour le ranger en mémoire, on lui allouera un "bloc" repéré par un "handle" (voir l'article sur la gestion de la mémoire).
Tout le GS (comme le Mac) est construit sur ce principe. Pour chaque besoin, Apple a défini des structures particulières, et a créé des outils destinés à les manipuler.
Par exemple, *QuickDraw* utilise *Cursor* (toutes les caractéristiques et le dessin du curseur), *Font* (la police de caractères, avec ses données accessoires *FontGlobalsRecord, FontID, FontInfoRecord, RomFontRec*), *GrafPort* (la structure de base de tout le système graphique), *LocInfo, PaintParam, PenState, ColorTable, QDProcs* (les procédures diverses de dessin), etc.
Lors de l'appel à l'une des fonctions des outils du GS, on procède en général à la manipulation de l'un des éléments de ces données structurées, ou du bloc entier. Les "ressources", en tant que structures de données, seront gérées de la même manière, avec l'aide d'un outil spécialement conçu pour cela: "Resource Manager".



Types standard et non-standard

Une ressource est une collection de données rangées dans un format défini (c'est la définition officielle), ce format étant laissé à l'initiative de l'utilisateur. Une ressource est désignée par deux numéros: le *type* (déterminant le format), et l'*ID* (numéro de la ressource dans le type).

Apple a défini des types de **ressources standard** utilisables par les outils du GS. Il n'y en a actuellement qu'un petit nombre, mais ils permettent déjà de faire beaucoup de choses. Vous trouverez ainsi parmi les types standard: les chaînes de caractères (type Pascal, ou type C); les blocs de texte, de longueur quelconque; les blocs des paramètres d'un menu, d'un item de menu, d'une fenêtre; le texte d'une fenêtre d'alerte; tous les blocs de paramètres des contrôles; des icônes; des dessins; des son digitalisés, etc.
L'emploi de ces ressources standard avec les nouveaux outils permet des simplifications spectaculaires de l'écriture des programmes. Ainsi, le lancement et la fermeture des outils, ou la création d'items de menus et de fenêtres, se font désormais en quelques minutes!

Les **ressources non-standard** peuvent contenir tout ce que vous voulez. Vous pouvez créer n'importe quel type de ressource dont vous pouvez avoir besoin (ne dépassez pas 64 Ko). A vous de définir le format de sa structure (sous forme de template, ou de "record" comme en Pascal); ensuite vous lui donnez un numéro de type de votre choix (pas un numéro déjà pris par une ressource standard). Vous mettez ce que vous voulez dedans, et vous pouvez ensuite confier le colis à ResMgr qui, en bon "chef de consigne de la gare", vous rangera votre bien dans une case, le couvera de ses soins attentifs jusqu'à ce que vous lui redemandiez.
Nous vous conseillons toutefois de ne créer vos propres ressources que quand vous aurez bien compris l'utilité et le mode d'emploi des ressources standard.

La gestion des ressources

Pour la gestion des ressources sur disque et en mémoire, l'outil "Resource Manager" se charge de tout. Vous n'avez pas du tout besoin de savoir de quelle façon les ressources sont rangées dans le fichier. ResMgr construit et tient à jour, dans le fichier ressources, un "carnet d'adresses" (*resmap*) dans lequel il note, dans l'ordre des types et dans l'ordre des ID pour chaque type, pour chaque ressource: le type, l'ID, son adresse dans le disque, ses attributs de stockage en mémoire, sa longueur et éventuellement un handle sur son adresse dans la mémoire.
A l'ouverture d'un fichier ressources, ResMgr charge en mémoire ce carnet d'adresses. Il en assurera la remise à jour permanente. A la fermeture du fichier, après avoir recopié les modifications de ressources dans le disque, il réécrira ce carnet, très souvent à un endroit différent du fichier. Les ressources elles-mêmes sont écrites un peu partout dans le fichier, là où il y a de la place. Dans le disque, ResMgr conserve aussi tout ce qui lui est utile pour disposer de l'espace libre, ici, et là, dans le fichier ressources. Il essaie d'utiliser au mieux les "trous" dans les blocs, trouve un endroit pour réécrire une ressource qu'on a agrandie, raye une ressource supprimée bref fait le ménage et range son coin, puis remet au net son carnet d'adresses. Laissons ResMgr faire son travail, et faisons-lui confiance.

ResMgr peut **ouvrir plusieurs fichiers successivement**, les laisser ouverts et rechercher une ressource donnée dans l'un ou l'autre de ces fichiers. Il y a un "ordre de recherche" fonctionnant comme une pile: la recherche commence par le dernier fichier ouvert, qu'on appelle le fichier "courant", puis l'avant-dernier, et ainsi de suite jusqu'au dernier, qui par principe est le fichier du système. ResMgr s'arrête quand il a trouvé une ressource dont le type et l'ID correspondent à ce qui lui est demandé.

Nous avons fait connaissance avec les ressources. Dans un prochain article, nous aborderons de façon plus précise la façon dont Resource Manager les gère, et comment nous pouvons utiliser les outils qu'il met à notre disposition. Mais d'ores et déjà, vous pouvez aller plus loin avec la documentation de ***ResDoctor***, incluse sur la disquette. ■

GE80: v2.0 ?

Quelques bugs encore dans le programme GE80 de ToolBox-Mag 4, et une erreur classique, dans le fichier de démonstration, concernant l'insertion de routines machine entre Basic System et ses buffers (merci à Yvan Kœnig de l'avoir signalée).

L'auteur, Mr Régis Mange, travaille à une nouvelle version.

Restez branchés sur ToolBox-Mag.

JYB

Introduction à la programmation de l'Apple II GS :

Du Basic au Pascal, 2º partie: les déclarations

Jean Destelle

Nous avons déjà pris un premier contact avec le langage Pascal. Nous allons maintenant avancer un peu plus, toujours dans l'esprit d'apprendre à programmer notre GS, à l'aide des outils de la Toolbox.

Le "langage" de la Toolbox

Le GS dispose déjà de son propre langage, avec la Toolbox. Tous les noms des fonctions, des constantes et des paramètres utilisés par les outils sont universellement employés, quel que soit le langage de programmation : Assembleur, C, ou Pascal. Il est facile à tout programmeur de lire et de comprendre les listings écrits dans l'un ou l'autre des langages, dès qu'ils utilisent la Toolbox.

Et cela représente un sérieux vocabulaire ! Pour vous en convaincre, feuilletez les listings reproduisant les units d'interface de votre TML Pascal II: 150 pages uniquement de "déclarations préalables", déjà établies afin de nous faire gagner du temps. Tous ces noms utilisés pour désigner les éléments de la boîte à outils sont autant de mots à votre disposition.

Lorsque vous programmerez, vous ne cesserez de vous y référer. Ce sera pour vous l'équivalent d'un dictionnaire orthographique. Beaucoup de ces mots proviennent, sans modification, des outils du Mac. De sorte qu'en apprenant à programmer votre GS, vous apprenez en fait aussi à programmer le cousin.

L'importance des déclarations

Nous avons déjà signalé l'importance de la question des déclarations préalables, qui est la principale difficulté du Pascal. Le langage Pascal, conçu pour pouvoir s'adapter à toutes situations, dispose de possibilités extrêmement étendues, régies par des règles très générales, mais fort abstraites. La pratique montre cependant qu'on emploie principalement, sur le GS, un nombre restreint de concepts avec lesquels il suffira de se familiariser.

En Applesoft, nous n'avons pas à déclarer une variable ou une constante avant de nous en servir. Il suffira d'écrire, par exemple *A = 40* ou *D$ = CHR$(4)* pour que la valeur 40 soit rangée en mémoire à un emplacement repéré par le nom de variable *A*, et que le caractère de code ASCII 4 soit, de la même façon, logé quelque part en mémoire, et soit ensuite désigné par le nom *D$*.

Nous avons ainsi procédé à la fois à la *déclaration* de la variable et à l'*assignation de sa valeur*. Seul le signe $ nous indique qu'il s'agit d'une variable de type "alphanumérique". L'absence de suffixe au nom de la variable *A* indique qu'il s'agit d'un nombre réel.

L'Applesoft a très peu de "types" différents: les nombres réels, les nombres entiers, et les chaînes de caractères, puis, comme types de données structurées, les tableaux de valeurs de ces types de base.

Peut-être vous êtes-vous déjà rendu compte des limitations de ce procédé: dès qu'on est amené à traiter de données un peu complexes, on est obligé de créer soi-même des structures spécifiques et d'en organiser la manipulation.

Notez, pour vous consoler, qu'en Assembleur, on est encore plus mal logé. On ne peut se référer qu'à des zones préalablement définies en mémoire. Seule, l'adresse de la zone de mémoire peut être prise en référence, sans aucune indication du format de son contenu. Il faut énormément de méthode pour s'y retrouver.

Le Pascal, tout au contraire, a été conçu pour s'adapter à la manipulation de tous les genres de données possibles et imaginables, à la seule condition d'indiquer avec précision, au préalable, les règles adoptées, cela au moyen de "déclarations".

Pascal est très pointilleux. **La plupart de vos ennuis en Pascal viendront d'une compréhension insuffisante de la terminologie des déclarations et de l'emploi des types.** Nous devons donc, avant d'aller plus loin, aborder un minimum de notions théoriques concernant le langage.

Les mots réservés du Pascal

Comme toute langue, le Pascal est composé de "mots", appelés ici *tokens*. Au moment de la compilation, le Pascal doit effectuer la reconnaissance de ces mots. Il les classe alors en plusieurs catégories:

1. Les séparateurs et opérateurs, composés d'un caractère (+ - / * = < > [ ] . , ( ) : ; ^ @ { }), ou de deux caractères (<> <= >= := .. (* *))

2. Des mots-clés, mots réservés ne devant pas être utilisés pour autre chose que leur emploi précis:

- mots-clés réservés aux déclarations: **const / var / type / array / record / set / file / label / packed of**
- mots-clés réservés au déroulement du programme:

if ... then ... else	**Case of ... otherwise**
repeat ... until	**While ... do**
for ... to	**for ... downto**
goto	**with ... do**

- mots-clés réservés à la structuration du programme: **procedure / function / program / begin / end / implementation / interface**

▶

- mots-clés servant d'opérateurs: **and / or / div / mod / in / not**
- mot-clé spécial désignant une valeur nulle: **Nil**

3. Des directives. Ce sont des mots-clés qui ne sont pas propres au Pascal en général, mais à votre compilateur. TML Pascal II reconnaît les directives suivantes: **EXTERNAL, FORWARD, INLINE, TOOL**

4. Des identificateurs. Ce sont les noms qui sont donnés aux constantes, aux types, aux variables, aux procédures, aux programmes, aux units, aux paramètres, etc, au moment de leur déclaration. Nous utiliserons souvent "nom d'identification" ou tout simplement "nom", pour désigner la même chose.
Beaucoup d'identificateurs sont déjà définis: certains désignent des procédures ou fonctions standard du Pascal; d'autres ont été déclarés dans les units d'interface et désignent des procédures, fonctions, constantes, paramètres, des divers outils du GS. Les autres identificateurs, nous les créerons nous-mêmes au fur et à mesure des besoins.
Chaque nom d'identification est une chaîne de 255 caractères maximum, composée de lettres, de chiffres et éventuellement du caractère '_' (souligné), parfois employé pour remplacer l'espace qui est interdit.
Tous les caractères sont reconnus (contrairement au Basic dans lequel seuls les deux premiers caractères sont significatifs). Le premier caractère est forcément une lettre. Le compilateur ne fait pas de différence entre majuscules et minuscules.
Exemples de noms d'identification choisis dans la Toolbox: *moveTo; NewHandle; GetCtlHandleFromID;*
Quelques autres exemples de noms possibles: *B672; a3; LeNomDuProgrammeur; la_procedure_2;*

NB: Depuis quelque temps, Apple tend à ne plus utiliser "_" comme séparateur, mais à insérer des majuscules dans les noms. L'habitude s'installe aussi de faire figurer devant chaque nom une ou deux lettres de reconnaissance indiquant la provenance de la déclaration: *gwindow1, gmemoryID, gMainEvent,* désignent des variables globales; *kAppleMenuRef, kControlID,* désignent des constantes. *TEGetSelectionStyle, TEKill, TEPaste,* désignent des fonctions de l'outil TextEdit, etc.
On a beaucoup coupé-collé pour programmer le GS, à partir d'exemples Apple ou TML faits pour cela, et les noms anglais sont restés. Employer des noms français est cependant souhaitable: ils se distinguent ainsi des noms employés par Apple dans la Toolbox.

5. Des nombres. Le compilateur reconnaîtra un mot comme étant un nombre s'il est exclusivement composé de chiffres et de quelques caractères autorisés, reconnaissables sur les exemples suivants:
43; +6744; -17.6; 12E4; 8.53206; 123E-7; $14FC
Ces nombres pourront être utilisés comme des valeurs de constantes ou de variables appartenant à des types divers. Cela n'est guère différent du Basic, sauf en ce qui concerne les nombres hexadécimaux qui, précédés du signe $, sont reconnus par le Pascal.

6. Des étiquettes (labels). Une étiquette, c'est un groupe de quatre chiffres au maximum (de 0 à 9999). Il servira à définir un point d'entrée dans le programme, à la suite d'une instruction *goto*. Absolument comme le numéro de ligne qui suit, dans le Basic, une instruction *Goto*.
En Pascal, une étiquette n'est pas un nombre. On ne peut donc pas faire d'opération arithmétique sur une étiquette.Vous n'emploierez que très rarement des étiquettes.

7. Des chaînes de caractères (strings). Nous en avons déjà parlé. Elles sont encadrées par le signe ' (apostrophe). Exemples: *'Pascal'; 'PARIS'; 'C''est bien'*. Si on veut utiliser une apostrophe dans une chaîne, il faut la doubler (*'C''est bien'*). La chaîne pourra être utilisée comme une constante ou une variable.

Les déclarations du Pascal

Le principe, déjà évoqué, est le suivant: avant d'employer un identificateur, il faut en avoir fait la déclaration, c'est-à-dire avoir décrit, suivant une règle précise, ce que désignera ce nouveau nom. Ainsi le compilateur pourra par la suite le traiter comme il convient. Voici un exemple de déclaration de variable: *var laChaine: str255;*
Le mot *var* indique qu'une ou plusieurs déclarations de variables vont suivre; *laChaine* est l'identificateur de la nouvelle variable. Après le séparateur *:*, le nom *str255* est l'identificateur du type de la variable.
Pour que cette déclaration soit valable, il faut que le type *str255* ait été auparavant lui-même déclaré. Vous retrouverez cette déclaration parmi celles de l'unit "Types" du TML II:

```
type    String255 = string[255];
        str255 = String255;
```

Le mot *type* précède obligatoirement les déclarations des types. Ici, plusieurs formes différentes ont été définies pour le même type: ce type particulier est extrêmement courant, et les programmeurs sont habitués à différentes désignations.
Cet exemple montre comment tout type doit être défini, à partir d'un autre type déjà existant. Au départ, Pascal ne connaît qu'un très petit nombre de types. Tous les autres sont construits au fur et à mesure à partir de là, au moyen de déclarations successives, ce qui ouvre des possibilités immenses.

Les cinq genres de déclarations

Pascal reconnaît cinq "sections" de déclarations: déclarations d'étiquettes (labels), de constantes, de types, de variables, de fonctions et procédures, avec une syntaxe bien définie. Nous nous contenterons d'exemples usuels.

1. Déclaration des étiquettes.

Mot-clé: *label*. Séparateur:
Exemples: *label 3;* (un seul label déclaré)
label 14,285,12; (trois labels)
Notez, à la fin de chaque déclaration, comme à la fin de toute ligne d'instruction Pascal, le séparateur *;*. Remarquez également qu'on peut déclarer successivement plusieurs étiquettes, en les séparant par des virgules. Cette facilité se rencontrera dans la plupart des déclarations.

2. Déclaration de constantes

Mot-clé: *const*. Séparateur: =
Une constante peut être soit un nombre, soit une chaîne de caractères, soit une autre constante désignée par son identificateur. Une fois la valeur attribuée, il n'est plus possible d'en changer.
Syntaxe: *identificateur = valeur;*
Exemple: *const Pi = 3.14159;*

▶

refIsHandle = $0001;
kString1 = 'Pascal';

Un très grand nombre de constantes ont déjà été définies par Apple pour les outils de la Toolbox. Vous les trouverez dans les units d'interface.

3. Déclaration de types

Mot-clé: ***type*****. Séparateur: =**
Syntaxe: *identificateur = le type de donnée;*
Exemple: *type byte = 0..255;*

4. Déclaration de variables

Mot-clé: ***var*****. Séparateur: :**
Syntaxe: *identificateur : type;*
Exemples: *var x, y, z : integer;*
laChaine : str255;
gMainEvent : EventRec;

5. Déclaration de fonctions et de procédures

Mots-clés: ***function, procedure*****. Séparateurs: ( ) ; :**
Exemple: *procedure Moveto (h: integer; v: integer);*
Cette procédure comporte deux "paramètres": *h* et *v*, qui sont déclarés, comme le seraient des variables, (en indiquant l'identificateur et le type) à l'intérieur de la parenthèse. Exemple:
function GetHandleSize (theHandle: handle): integer;

La fonction se déclare comme la procédure, mais se termine par l'indication du type du résultat qu'elle fournit, précédée du séparateur *:*.
Notez qu'il n'y a pas de point-virgule après la dernière déclaration de paramètre à l'intérieur de la parenthèse.

Vous trouverez de nombreux exemples de déclarations de fonctions et procédures dans les units d'interface. En cas de doute sur la syntaxe, piochez dans les listings de ces units.

Blocs de programme et portée d'une variable

Pascal est un langage modulaire: tout programme écrit se décompose en *blocs*. Chaque bloc doit est composé de deux parties essentielles: les déclarations et les instructions.

Les **déclarations** apparaissent généralement dans l'ordre logique suivant: étiquettes, puis constantes, puis types, puis variables, enfin fonctions et procédures . La séquence des **instructions** commence par *begin* et se termine par *end*.

Dans un programme, les "blocs" peuvent s'imbriquer les uns dans les autres. Le programme lui-même constitue un "bloc", qui recouvre tout l'ensemble des autres blocs. C'est pourquoi les déclarations effectuées dans ce bloc portent le nom de "globales".
Chaque procédure ou fonction constitue aussi un "bloc", avec ses propres déclarations, qui sont "locales", et sa séquence d'instructions.
A l'intérieur du bloc d'une procédure, on peut très bien trouver d'autres blocs, par exemple des procédures ou fonctions dites "locales", utilisées seulement par la procédure principale.

Tous les identificateurs déclarés dans un bloc ont **une portée locale** limitée à ce bloc. Cela signifie qu'on ne peut s'y référer que de l'intérieur de ce bloc. Ces identificateurs n'ont pas de signification pour les procédures et fonctions extérieures au bloc.
De même, les déclarations faites en début de programme, appartenant au bloc du programme, ont une **portée globale**. Les identificateurs peuvent être alors utilisés d'un point quelconque du programme. On parle ainsi de variables ou constantes "globales" ou "locales". On peut employer le même nom d'identification pour des variables **locales** appartenant à des blocs différents.

Une erreur fréquente consiste à déclarer, comme nom de variable locale, un nom déjà précédemment employé pour déclarer une fonction globale. Prenez le soin d'utiliser, pour vos variables locales, toujours les mêmes noms, qui deviennent en quelque sorte des "variables à jeter".
Dans les programmes américains, on rencontre ainsi souvent le suffixe "temp" à la fin du nom d'une variable temporaire. Pour ma part, j'utilise des noms de variables locales très courts, comme "a,b,c,d,...,L, x,y,z" pour désigner des nombres et "s1,s2,s3..." pour désigner des chaînes. Pour des variables de types moins courants, je reprends souvent le nom du type précédé de un article "le", ou "la", ou par "mon" ou "ma". Exemples:
var a,b: integer; s1,s2: str255; leNameRecGS: NameRecGS;

Lorsque des noms d'identification ont été déclarés dans la section des déclarations globales d'une Unit, si l'Unit fait partie de la liste figurant à la suite du mot *uses* en début de programme, ils sont globaux pour le programme.

Les types standard

Le mot "type" définit le genre de données qui est désigné par un identificateur. Chaque variante du langage Pascal définit un nombre restreint de types standard. Ensuite les utilisateurs peuvent à leur tour définir un grand nombre d'autres types, en fonction des besoins.
Nous reprendrons ici les types standard du langage TML Pascal II, qui sont pratiquement les mêmes que ceux des autres variantes de Pascal GS. Ils sont classés en cinq catégories: les types simples, les types structurés, les types "chaînes", les types "pointeurs", les types "objets".

1. Les types simples

1.1. Types ordinaux standard.

Il y a quatre types standard de base, qui existent dans toutes les versions du Pascal: *Integer, LongInteger, Boolean, Character*.
• *Integer* désigne un nombre entier, positif ou négatif, pouvant être stocké en mémoire sur deux octets, compris entre -32 768 et +32 767. Souvent désigné par *word* (mot) dans d'autres langages et les ouvrages de référence Apple, c'est l'élément de base des nombres en Pascal. Notez qu'un nombre écrit sur un seul octet (*byte*) n'est pas un type standard .
• *LongInt* désigne un nombre entier long, positif ou négatif, pouvant être stocké sur 4 octets en mémoire. Sa valeur est comprise entre -2 147 483 648 et + 2 147 483 647.
Les fonctions *LoWrd* et *HiWrd*, que nous utiliserons souvent dans les boucles d'événements du GS pour prendre les renseignements fournis par Task- ▶

Master, permettent de "sortir" chaque moitié de ce longinteger: le *high word*, le mot de poids le plus élevé, et le *low word*, le mot de poids le plus bas.

• *Boolean*, donnée logique pouvant prendre soit la valeur *false* (faux), soit la valeur *true* (vrai). De même qu'en Basic, on peut convertir *true* et *false* en 1 et 0. Pour transformer la donnée logique en un type numérique, on utilise la fonction de conversion *ord(...)*: *ord(false)* vaut 0, et *ord(true)* vaut 1.

• *Char* désigne un seul caractère. Il peut être stocké en mémoire sur un seul octet, qui prend alors la valeur du code ASCII du caractère. Cette forme de stockage est réservée aux chaînes de caractères et aux tableaux et records "compactés". Dans les autres cas, le Pascal emploie deux octets de mémoire.

De la même façon que ci-dessus, on peut manipuler le code ASCII du caractère comme une donnée numérique en employant la fonction logique *ord(..)*; *ord('A')*, par exemple, désignera le nombre 65, code ASCII de 'A', ce qui correspond à la notation en Basic *ASC("A")*.

Comme en Basic, on trouve la fonction inverse: *Chr(..)* (mais sans le signe $) pour transformer un code ASCII en un caractère; *Chr(65)* représente la lettre 'A'.

1.2 Les types ajoutés

1. *Enumération:* c'est le contenu d'un ensemble, sous forme de liste énumérée dans un ordre défini. On appelle "ordinalité" le numéro d'ordre de l'élément dans la liste (en commençant par 0). Même si les objets énumérés ne sont pas quantifiables, leur valeur "ordinale" peut donner lieu à des comparaisons. L'élément peut être n'importe quoi. Par exemple:

couleur : (rouge, vert, bleu, blanc, jaune);

Cette déclaration concerne un type dont l'identificateur est *couleur*, et qui est défini par ses composants possibles: à la suite de la déclaration, seules les cinq couleurs désignées pourront appartenir à ce type.

L'énumération de l'ensemble porte aussi parfois le qualificatif de "scalaire", si l'ensemble peut être classé dans un ordre défini.

Certaines fonctions du Pascal permettront d'opérer sur les valeurs ordinales. Ainsi, dans cet exemple, *ord(rouge)* désigne le numéro d'ordre du rouge, c'est-à-dire 0. On pourra écrire: *ord(vert) < ord(blanc)*; cette expression aura une valeur booléenne *true*.

Il existe aussi les fonctions *pred( )* (précédent élément) et *succ( )*, (élément suivant). Ainsi *pred(jaune) = blanc* et *succ(vert) = bleu*.

Je vous laisse le soin d'imaginer tous les développements possibles de ce principe. En général, une énumération désignera plutôt un ensemble de nombres, comme: 0,1,2,3,5.

2. *Intervalle* (SubRange). Un intervalle est un groupe d'éléments successifs prélevés dans une énumération. On le désigne par deux bornes: le premier et le dernier élément, séparés par deux points successifs. Exemples:

1 .. 100 ; 128 .. 127; vert .. blanc

Les deux bornes doivent être écrites dans le bon ordre: la valeur ordinale de la première doit être inférieure à la valeur ordinale de la seconde.

3. *Real* (Nombre réel). Les nombres réels peuvent s'écrire sous leur forme numérique habituelle, par exemple 3.14159 (notez que le point remplace la virgule décimale en Pascal comme en Basic) ou en "notation scientifique", toujours comme en Applesoft.

TML Pascal II utilise quatre types différents de réels:

• le type *single* ou *real* qui utilise quatre octets de mémoire, et correspond à un nombre compris entre 1.4E-45 et 3.4E38, ce qui correspond à des nombres comportant 7 ou 8 chiffres significatifs. Cette précision est largement suffisante pour les cas courants de calcul technique ou scientifique.

• le type *double*, qui demande huit octets en mémoire, et fournit des nombres jusqu'à 15 et 16 chiffres significatifs.

• le type *extended*, stocké sur dix octets de mémoire, permet des nombres de 19 à 20 chiffres significatifs.

• enfin, le type *Comp* ou *Computational*, stocké sur huit octets, et fait pour l'emploi des grands nombres en système décimal.

Vous trouverez tous les renseignements utiles sur ces nombres dans la documentation officielle: *Apple Numerics Manual*, qui traite du *SANE* (Standard Apple Numerics Environment).

Remarquez que de très nombreux calculs peuvent se faire avec des nombres entiers de type *integer* ou *longinteger*, même des calculs comptables: il y a des astuces, par exemple travailler en nombres entiers de centimes, et procéder à des divisions par 100 au moment de l'affichage, par un simple traitement de la chaîne de caractères. Les calculs sont alors beaucoup plus faciles et plus rapides.

2. Les types structurés

Nous rentrons maintenant dans une partie importante pour la programmation du GS, car nous rencontrerons une foule de données structurées utilisées par la Toolbox. Surtout sous forme de *records*, qui peuvent prendre les aspects les plus variés.

Les types structurés du Pascal comprennent les tableaux (*arrays*), les *records*, les ensembles (*sets*), les *files* (mot anglais mal traduit par "fichier"; nous emploierons plutôt le mot français "file"), les chaînes (*strings*), les pointeurs (*pointers*).

Chacun de ces types peut exister sous sa forme normale ou sous forme "compactée" (*packed*). Quand la forme compactée est employée, les données qui devraient être stockées normalement sur un octet (comme les *bytes* ou les *characters*) n'occupent effectivement qu'un octet en mémoire. Sinon, chaque donnée occupe deux octets en mémoire. (Cette possibilité ne joue en fait que pour les tableaux et les Records. Déclarer un autre type structuré comme compacté ne change rien à son stockage en mémoire.)

2.1. Les Tableaux

Il y a une très grande analogie entre les tableaux du Basic et ceux du Pascal. Les tableaux du Basic devaient aussi être "déclarés" , en fait "dimensionnés" au moyen des instructions *DIM (...)*, afin de réserver leur emplacement en mémoire. Au lieu des parenthèses du Basic, le Pascal emploie des crochets. Mais pour le reste, la désignation est similaire.

En Pascal, on peut loger dans les tableaux des données extrêmement variées, comme des "records" complexes. Toutefois, toutes les données d'un tableau doivent avoir le même type. Le nombre possible d'éléments est fixé dans le libellé du type ▶

au moment de la déclaration de la variable. Exemple: *array [1 .. 100] of Real*: tableau de 1 à 100 nombres réels. Entre les crochets figure le type de l'"index", qui est forcément un type "ordinal".
Généralement, on trouve une énumération, ou, comme ici, un intervalle. Mais ce n'est pas forcé. On peut très bien employer un identificateur de type défini précédemment, par exemple:
packed array [couleur] of Boolean.
Ce tableau pourra comporter autant d'éléments que le type *couleur*.

On peut avoir bien évidemment des tableaux à plusieurs entrées. Les divers index sont alors déclarés, dans la déclaration de type, dans l'ordre souhaité, séparés par des virgules (comme en Basic). Ainsi *array [1..20, 1..10] of integer* représentera un tableau de nombres entiers, à double entrée, le premier index acceptant des valeurs de 1 à 20, et le second de 1 à 10.
L'utilisation des tableaux dont les index ne sont pas des nombres simples n'est pas toujours facile à interpréter dans les sources, aussi est-il bon d'en réserver l'emploi à des utilisateurs très confirmés.

Emploi des tableaux. Une fois qu'une variable de type tableau a été déclarée, on peut l'utiliser pour assigner des valeurs à ses éléments, et employer ces valeurs, comme toute autre variable. Prenons un exemple simple: déclarons une variable de type tableau: *var Age : array [1 .. 100] of integer;*
La désignation d'un des éléments du tableau se fera en faisant suivre l'identificateur du tableau par la valeur de l'index, entre crochets. Ainsi *Age [11]* désignera le onzième élément du tableau, dans la suite des instructions. On pourra écrire par exemple, pour "remplir" les cases du tableau:
Age[1] := 17; Age[2] := 21; Age[2] := 9; etc
Même principe pour utiliser les données du tableau:
if Age [a] > 17 then majeur := true;

Remarque: Nous ne pouvions, en Basic, traiter que les données élémentaires d'un tableau. En Pascal, **le tableau entier lui-même constitue une variable utilisable comme une autre**. On pourra ainsi la déplacer en bloc dans la mémoire. Ou bien en faire l'un des éléments d'une donnée structurée plus complexe, comme un *record*.

2.2 Les Records

La traduction du mot *record* par *enregistrement* restreint le sens de son emploi en Pascal. Nous préférerons conserver le mot anglais, dans l'acceptation que lui donne le Pascal: un ensemble fixe de données appelées *fields* (champs), les "champs" pouvant être de sens différents. Dans ce qui suit, nous utiliserons plus souvent le mot "élément" ou même "paramètre" pour désigner l'un des champs d'un record. En effet, dans la programmation du GS, la principale utilisation des *records* est la constitution de blocs de paramètres destinés aux fonctions des outils de la Toolbox.
Le *record* est une structure extrêmement souple, capable de s'adapter à bien des situations. Pour vous en rendre compte, parcourez les listings des units d'interface, et voyez combien de types de records différents ont été définis, beaucoup comportant même, comme élément, un ou plusieurs autres *records*. C'est également ce genre de structure que nous rencontrerons dans les formats des ressources qui ont été créés pour la manipulation de ces données.
Nous prendrons comme exemple l'un des records définis pour QuickDraw2: *LocInfo*, employé pour définir un GrafPort. Déclaration du type de record:
type LocInfo = Record
portSCB : integer;
ptrToPixImage : ptr;
width : integer;
boundsRect : rect;
end;
Dans cette déclaration, on trouve d'abord l'identifiant de la variable, *LocInfo;* puis, suivi du mot-clé *Record*, la liste des champs du *record*, chaque champ étant déclaré comme une variable ordinaire: *identificateur: type;* le mot-clé *end* indique la fin de la déclaration.

Les identificateurs de chacun des champs peuvent reprendre des noms déjà utilisés par ailleurs, car ils n'auront, aux yeux du compilateur, un sens précis que s'ils sont rattachés au nom du record lui-même. De ce fait de nombreux éléments de records différents portent le même nom de champ. Mais vous ne devez pas utiliser deux fois le même identificateur dans un même record, cela va de soi.
Notez que ce record fait appel à des éléments de types variés. Le dernier élément, *boundsRect*, qui désigne le rectangle limite du grafport, est lui-même un *record*, de type *rect* (rectangle), comportant quatre éléments, les positions de chacun des côtés.

Emploi du type. Ce type que nous venons de définir pourra être utilisé dans la déclaration d'une variable. Par exemple: *var le LocInfo: LocInfo;*
Cela signifie que nous désignerons par l'identificateur *leLocInfo*, une variable de type *LocInfo*. Pour le compilateur, ce sera un ordre de réservation pour un emplacement en mémoire, taillé sur mesure pour recevoir la totalité du record, soit 16 octets:

portSCB :	entier	2 octets
ptrToPixImage :	pointeur	4 octets
width :	entier	2 octets
boundsRect : rectangle (4 integers)		8 octets

Notez que ce travail de réservation en mémoire, le compilateur le fera avec ses propres moyens, sans que nous ayons à nous en occuper. Il a prévu des zones spéciales pour cela, différentes selon que cette variable est globale ou locale. S'il s'agit d'une variable locale, elle sera placée dans la pile, à titre très provisoire, et effacée dès qu'on n'en aura plus besoin.

Utilisation des données. On peut accéder à chaque champ du record en le désignant par l'identificateur du record, suivi d'un point, puis du nom de champ. Par exemple, *leLocInfo.PortSCB* désignera le champ *PortSCB* de notre record *leLocInfo*. On pourra utiliser cette locution comme le nom d'une variable ordinaire. De même *leLocInfo.Width* désignera le champ *Width*, et *leLocInfo.BoundsRect.top* désignera le champ *top* (ordonnée du côté haut du rectangle) du *BoundsRectangle* de *leLocInfo*.

Entrée des données. Nous disposons de deux moyens. Le premier consiste simplement à faire des "assignations" successives sur chacun des champs à valoriser. Par exemple:

leLocInfo.portSCB := getCurPort;
leLocInfo.ptrToPixImage := @MonImage;
leLocInfo.width := 640;
leLocInfo.BoundsRect := rectangle1;

Nous disposons aussi d'un moyen plus rapide d'écriture, grâce à l'instruction *with...do*:

```
with leLocInfo do begin
        portSCB := getCurPort;
        ptrToPixImage := @MonImage;
        width := 640;
        BoundsRect := rectangle1;
end;
```

With peut aussi être employé pour utiliser une donnée déjà valorisée. Par exemple:
with leLocInfo.BoundsRect do
if rectangle1.Top > 40 then TropGrand := true;
Remarquez qu'avec *with*, le nom du champ est débarrassé de l'identificateur du record. Alors, attention: si vous avez pris un nom qui est aussi celui d'une variable globale, le compilateur risque de ne pas savoir ce dont il s'agit.

Pour manipuler en bloc le record entier, on utilise simplement le nom du record comme une variable. En général, cette manipulation se fera à propos de la position en mémoire du record, et on la traitera plutôt en employant un *pointeur* sur le record, ou encore plus généralement au moyen d'un *handle* si nous avons nous-même défini le bloc de mémoire occupé par ce record.

2.3. Les Ensembles (Sets)

Ils sont bien moins employés dans la programmation du GS que les autres types structurés. Un type "ensemble", doit être déclaré sous la forme *set of* suivi d'un type ordinal (type de base). Par exemple: *set of 0..100, set of char.*

2.4. Les Files

Un type *file* est une suite linéaire de composants d'un même type. Il n'y a pas de nombre de composants défini dans la déclaration. L'emploi principal de ce type réside dans les "files de caractères". Sous forme compactée, une file de caractères prend le nom particulier de *Text*, forme très utilisée dans les instructions d'entrée et sortie du Pascal.
La déclaration du type se fait sous la forme *(packed) file of* suivi du nom du type des composants.
Il n'existe pas de moyen direct pour manipuler les composants d'une file. On doit écrire les instructions voulues pour trouver le composant.

2.5. Les Chaînes (strings)

Une chaîne est une suite de caractères représentés chacun par un octet, et dont le premier octet indique le nombre de caractères de la chaîne. Ce type de chaîne est communément appelé "Chaîne Pascal". C'est en fait une *packed file of char* un peu spéciale, qui bénéficie d'un mode d'accès direct pour chacun de ses composants. Le nombre d'octets est limité à 255 (comme en Basic).

Il existe d'autres types, non standard, de chaînes: par exemple la "chaîne C" (CString), qui ne comporte pas d'octet de longueur au début, mais un 0 à la fin, et la chaîne de type "étendu" ou GSString qui comporte au début un integer à la place de l'octet de longueur, ce qui ne limite plus sa longueur à 255. Ce type de chaîne est utilisé communément par les nouveaux outils du GSOS.

On peut déclarer des types de chaînes Pascal de moins de 255 caractères de longueur. Par exemple, si on veut manipuler des noms de fichiers qui ne dépasseront jamais une quinzaine de caractères, il n'est pas nécessaire de réserver l'espace pour 255 caractères, ce qui encombre la pile.

Exemple de déclaration de type chaîne:
type str32 = string [32];
Le nombre de caractères souhaités se trouve à droite, entre crochets.

Emploi des chaînes. On déclarera les variables chaînes comme toute autre variable. Par exemple:
var laChaine : str255;
Pour donner à une variable chaîne sa valeur, on fera une assignation , et le texte à écrire sera inclus entre des apostrophes, comme pour une constante-chaîne.
Exemple: *laChaine := 'Pascal';*

Pour accéder à chaque élément d'une chaîne Pascal, on dispose d'un moyen très commode. Le mode d'écriture est le même que si la chaîne était un *packed array of char*. Ainsi:
laChaine[1] désigne le 1er caractère 'P',
laChaine[2] désigne le 2ème caractère 'a',
laChaine[3] désigne le 3ème caractère 's',
... etc...
De plus le code ASCII du caractère *laChaine[0]* désigne la longueur de la chaîne. On écrira:
lalongueur := ord(laChaine[0]);
laChaine[0] désigne le premier octet du bloc de mémoire occupé par *laChaine*. La valeur de la longueur sera 6. C'est le *nombre de caractères* et non le *nombre d'octets occupés en mémoire par la chaîne*, qui est de 7.
Si nous voulons accéder à ce bloc au moyen d'un pointeur, nous écrirons:
lepointeur := @laChaine;
Ce pointeur sera l'adresse du caractère zéro de la chaîne en mémoire. Si nous voulions (par exemple pour transférer cette donnée vers un buffer d'imprimante) pointer sur le premier caractère de la chaîne, il suffirait d'écrire:
lepointeur := @laChaine[1];
Vous trouverez des explications sur les pointeurs et handles dans l'article sur la gestion de la mémoire de cette série.

Autres fonctions agissant sur les chaînes. Puisque nous sommes dans les chaînes, profitons-en pour citer quelques fonctions très utiles permettant de les manipuler.
• *Length(str)* fournit un integer, le nombre de caractères de la chaîne (comme en Basic).
• *Concat(str1,str2,...)* fournit une chaîne qui est la concaténation (c'est-à-dire la mise bout à bout) de plusieurs chaînes. Equivalent de *A$ = A1$ + A2$* en Applesoft.
• *Copy(source, index, count)* retourne *count* caractères de la chaîne *source*, prélevés à partir du caractère de numéro *index*. Identique au *MID$(..)* du Basic.
• *Pos ('mot', str)* recherche la petite chaîne *mot* à l'intérieur de la chaîne *str*, et retourne le numéro d'ordre du premier caractère de *mot*. Cette fonction n'existait pas en Applesoft.

▶

Mentionnons enfin deux procédures également inconnues en Applesoft:
• *Delete(str, index, count)* supprime *count* caractères de la chaîne *str* à partir du caractère du numéro *index.*
• *Insert(source, dest, index)* ajoute dans la chaîne *dest,* la chaîne *source,* en l'insérant à partir du caractère numéro *index.*

2.6. Les types Pointeurs

Une variable de type pointeur contient l'adresse en mémoire d'une variable dynamique (c'est à dire susceptible d'occuper des positions différentes en mémoire). Le type de la variable "pointée" s'appelle le "type de base". On peut ainsi construire, en Pascal, des types de pointeurs très variés, correspondant chacun à un type de base déterminé. Car un pointeur ne sert pas à grand-chose si on ne sait pas ce qu'il désigne. Le Pascal utilise un moyen simple (peut-être trop simple) pour prélever une donnée à l'adresse indiquée par un pointeur. De sorte que si on ne précise pas, il est capable de lire n'importe quoi. Si le pointeur a un type bien précis, le Pascal prélèvera la donnée du format précisé par le type de base de ce pointeur.
Exemple de déclaration de type pointeur:

```
        type string255ptr = ^string255;
```

Pour "fabriquer" un type de pointeur, nous avons ajouté, devant le nom du type de base, un "circonflexe" qui transforme ce nom en celui d'un pointeur. Ce circonflexe se lit "pointeur vers". Il a la même fonction que le signe @, qui permet de créer un pointeur vers une variable Pascal existante. Nous pourrions écrire ceci:

```
var     lePointeur:   string255ptr;
        laChaine:     string255;
            begin
            ......
            lePointeur     := @laChaine;
            ......
            end;
```

Utilisation des pointeurs. Le signe ^ (circonflexe) placé devant un identificateur de variable signifie "pointeur vers" la position, en mémoire, de cette variable. Plus précisément, il s'agit de l'adresse en mémoire du premier octet du bloc occupé par cette variable.
Le même signe placé à la fin de l'identificateur d'une variable de type pointeur désigne le contenu de la variable.
Dans l'exemple précédent, *lePointeur^* désignera le contenu du bloc de mémoire pointé par la variable *lePointeur,* c'est-à-dire la variable *laChaine.* Vous comprenez pourquoi il était nécessaire que le type de la variable *lepointeur* soit précisé, si on veut que le contenu du bloc qu'il désigne le soit aussi.

Dans le GS, on utilise beaucoup les *handles* qui sont *des pointeurs vers d'autres pointeurs.* Les *handles* restent écrits à des endroits fixes de la mémoire. Ils contiennent l'adresse (c'est-à-dire le pointeur) actuelle du bloc de mémoire correspondant. Vous rencontrerez donc parfois l'emploi de deux circonflexes successifs: par exemple *leHandle^^* désignera la donnée se trouvant dans le bloc de mémoire (dynamique, donc mobile) géré par *leHandle.* Il existe ainsi toute une gamme de *types de handles,* pratiquement un par type de donnée de base.

La constante Nil. La constante *Nil* est un pointeur "universel" de valeur nulle, que le compilateur du Pascal accepte à la place de n'importe quel autre type de pointeur (ou de handle puisque un handle est un pointeur).
Elle est très employée dans la programmation du GS, car Apple lui donne souvent une signification particulière, en tant que paramètre dans l'appel d'une fonction de la Toolbox.

Compatibilité des types. Pour qu'une instruction puisse être compilée, il faut que les types employées par les variables utilisées soient "compatibles". Par exemple, une "assignation", qui fixe la valeur d'une variable, doit comporter, de chaque côté du signe := des éléments de types "compatibles". Dans *variable1 := variable2,* le type de la *variable2* est exactement le même que celui de la *variable1,* il y a "identité" des types. Donc ils sont compatibles.
S'il s'agit de types très usuels comme des nombres, il peut y avoir de nombreux cas de compatibilité. Par exemple si *variable1* est un *longinteger,* et *variable2* un *integer,* il n'y aura aucun problème. De même pour les chaînes si *variable1* est de type *str255,* et si *variable2* est de type *str32.*

Les règles générales de compatibilité sont assez indigestes. Mais le compilateur est extrêmement chatouilleux sur cette question, et annonce une erreur chaque fois qu'il y a incompatibilité: la pratique vous apprendra donc très vite les bévues à éviter.

Changement de type (*Cast*). Dans la plupart des cas, vous trouverez facilement la cause de votre erreur, et vous la corrigerez. Parfois, il vous suffira d'opérer un "changement de type" (cast) , permis par certaines versions de Pascal, dont le TML II.
Par exemple, si le compilateur attend comme paramètre d'une fonction un pointeur de type ordinaire, et si vous lui fournissez un pointeur nommé *monPointeur* de type *String255ptr,* il vous le signalera en stoppant la compilation à cet endroit et en vous envoyant un message d'erreur explicite. Il vous suffira d'opérer un changement de type en écrivant *ptr(MonPointeur)* au lieu de *MonPointeur.* Vous aurez ainsi informé le compilateur que vous considérez, dans cette instruction, le pointeur *monPointeur* comme un pointeur de type ordinaire *ptr.* Et tout rentrera dans l'ordre.
Ce changement de type n'est possible que si les emplacements en mémoire des données de l'ancien type et du nouveau sont de même dimension.
Ce procédé vous sera extrêmement utile, car de nombreux outils de la Toolbox ont été déclarés dans les interfaces, avec des paramètres d'entrée qui sont des pointeurs ou des handles de type très particuliers, alors que certains autres outils, comme MemoryMgr, travaillent le plus souvent avec des pointeurs et des handles de type commun. Il est donc souvent nécessaire d'avoir recours à des changements de types.
Le *cast* est possible en TML Pascal II sur tous les types, sauf les ensembles (sets).

Nous en resterons là pour cette fois. Sachez que le plus difficile est désormais derrière vous!

■